Zbiór zadań z programowania w języku C/C++ cz. 2

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf”.
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

Uniwersytet Marii Curie-Skłodowskiej
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Zbiór zadań z programowania w języku C/C++ cz. 2

Jacek Krzaczkowski

Lublin 2012

Instytut Informatyki UMCS
Lublin 2012

Jacek Krzaczkowski

Zbiór zadań z programowania w języku C/C++ cz. 2

Recenzent: Grzegorz Matecki

Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl.

Wydawca
Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikołajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk
FIGARO Group Sp. z o.o. z siedziba w Rykach
ul. Warszawska 10
08-500 Ryki
www: www.figaro.pl

ISBN: 978-83-62773-23-7

Spis treści

Przedmowa

Niniejszy skrypt jest drugą częścią zbioru zadań pt. „Zadania z programowania C/C++". Pierwsza część, zawierająca zadania z programowania strukturalnego, została wydana przez Instytut Informatyki UMCS w 2011r. Po roku autor oddaje w ręce czytelnika drugą część zbioru poświęconą programowaniu obiektowemu i zaawansowanym technikom języka C++, takim jak szablony i wyjątki. W przeciwieństwie do pierwszej części przeznaczonej w równym stopniu dla uczących się języka C jak i C++, druga część zbioru zadań dotyczy niemal w całości języka C++. Wyjątkiem jest kilka zadań w rozdziale 7 „Wyjątki" ilustrujących sposoby radzenia sobie w języku C z błędami pojawiającymi się w trakcie działania programu.

Język C++ jest jednym z najpopularniejszych współczesnych języków programowania. Umożliwia on programowanie zgodne z różnymi paradygmatami programowania np: programowanie strukturalne, obiektowe czy generyczne. Ta uniwersalność języka C++ powoduje, że niełatwo jest uwzględnić w książce wszystkie aspekty związane z programowaniem w nim. W wielu rozdziałach autor musiał dokonać selekcji poruszanych zagadnień, tak żeby z jednej strony zbiór nie rozrósł się zbytnio, a z drugiej strony żeby móc dać więcej niż po jednym zadaniu dotyczącym najważniejszych kwestii.

Na końcu książki czytelnik znajdzie rozwiązania części zadań. Znajdują się tam rozwiązania zadań reprezentatywnych dla poszczególnych zagadnień, jak również zadań z różnych powodów ciekawych.

Układając zadania autor starał się, żeby dotyczyły one realnych problemów programistycznych, a ich rozwiązania były na tyle, na ile to możliwie, krótkie. Nie było to jednak proste, szczególnie w przypadku zadań mających za zadanie pomóc w opanowaniu bardziej zaawansowanych tematów. Przy pisaniu tego skryptu autor wielokrotnie stawał przed wyborem, czy umieścić w zbiorze oczywiste w danym kontekście zadanie o długim rozwiązaniu, czy zadanie może nie tak naturalne, ale za to o krótkim i ciekawym rozwiązaniu. Autor ma nadzieję, że w większości przypadków dokonał właściwego wyboru.

W trakcie pisania skryptu został opublikowany od dawna oczekiwany nowy standard języka C++. Autor stanął więc przed wyborem, czy skrypt pisać pod kątem starego czy nowego standardu. Ze względu na fakt, że znaczna część nowego standardu nie została jeszcze zaimplementowana w najpopularniejszych kompilatorach, niniejszy zbiór skupia się na starym standardzie. Jedyne odstępstwo zostało zrobione w przypadku rozdziału dotyczącego STL-a, w którym to rozdziale została umieszczona pewna liczbę zadań pozwalających przećwiczyć użycie nowych elementów biblioteki STL. Zadania te zostały specjalnie oznaczone.

Przy niektórych zadaniach znajdują się różne oznaczenia. Poniżej znajdują się wyjaśnienia używanych oznaczeń:

***** trudne zadanie,

r zadanie rozwiązane w ostatnim rozdziale,
! zadanie, którego rozwiązanie z różnych powodów jest szczególnie interesujące,
C zadanie, które można rozwiązać także w języku C.
C++11 zadanie pozwalające przećwiczyć użycie elementów wprowadzonych w nowym standardzie języka C++.

Rozdział 1

Klasy, obiekty, dziedziczenie

1.1. Klasy, obiekty

1.1 **(r)** Napisz klasę `poczta` zawierającą publiczne pola do przechowywania danych wiadomości przesłanej pocztą elektroniczną: `nadawca`, `odbiorca`, `temat` i `tresc`.

1.2 **(r)** Napisz funkcję `wypisz`, która jako argument otrzymuje obiekt typu `poczta` z zadania 1.1 i wypisuje na standardowym wyjściu wartości pól otrzymanego w argumencie obiektu.

1.3 **(r)** Napisz funkcję `wczytaj`, która jako argument otrzymuje referencję do obiektu typu `poczta` z zadania 1.1 i wczytuje ze standardowego wejściu wartości pól obiektu, do którego referencję otrzymała w argumencie.

1.4 **(r)** Do klasy `poczta` z zadania 1.1 dopisz metody wczytujące i wypisujące przechowywane dane.

1.5 **(r,!)** Zdefiniuj strukturę `poczta2` o takich samych polach publicznych jak klasa `poczta` z zadania 1.1. Napisz rozwiązania zadań od 1.2 do 1.4 w wersji dla struktury `poczta2`.

1.6 Napisz klasę `ksiazka` zawierającą publiczne pola `tytul`, `autor`, `wydawca`.

1.7 Napisz funkcję `wypisz`, która jako argument otrzymuje obiekt typu `ksiazka` z zadania 1.6 i wypisuje na standardowym wyjściu wartości pól otrzymanego w argumencie obiektu.

1.8 Napisz funkcję `wczytaj`, która jako argument otrzymuje referencję do obiektu typu `poczta` z zadania 1.6 i wczytuje ze standardowego wejściu wartości pól obiektu, do którego referencję otrzymała w argumencie.

1.9 Do klasy `ksiazka` z zadania 1.6 dopisz metody wczytujące i wypisujące pola obiektu.

1.10 Napisz klasę `trojkat` zawierającą:
- publiczne pola `wysokosc` i `podstawa`,
- publiczne metody służące do wczytywania ze standardowego wejścia i wypisywania na standardowym wyjściu wartości pól obiektu,
- publiczną metodę `pole` zwracającą jako wartość pole trójkąta o wymiarach przechowywanych w obiekcie.

1.11 Napisz funkcję, która dostaje w argumentach dwa obiekty typu `trojkat` z zadania 1.10 i wypisuje na standardowym wyjściu wymiary tego spośród trójkątów otrzymanych w argumentach, który ma większe pole.

1.12 Napisz funkcję, która dostaje w argumentach tablicę obiektów typu `trojkat` z zadania 1.10 oraz jej rozmiar i wypisuje na standardowym wyjsciu wymiary tego spośród trójkątów otrzymanych w argumentach, który ma większe pole.

1.13 Napisz klasę `funkcja` służącą do operowania na funkcjach liniowych jednej zmiennej. Klasa `funkcja` powinna posiadać publiczne pola `a` i `b`

i publiczną metodę `wartosc`, która dla podanego argumentu `x` zwraca wartość funkcji obliczoną ze wzoru `f(x)=a*x+b`.

1.14 Napisz klasę `funkcja_kw` służącą do operowania na funkcjach kwadratowych jednej zmiennej. Klasa `funkcja` powinna posiadać publiczne pola `a`, `b` i `c` oraz publiczne metody:
- `wartosc` zwracającą wartość funkcji w podanym jako argument metody punkcie `x` obliczoną ze wzoru `f(x)=a*x*x+b*x+c`,
- `zero` zwracającą `true`, jeżeli przechowywana funkcja ma rzeczywiste miejsce zerowe oraz `false` w przeciwnym wypadku.

1.15 **(r)** Napisz klasę `liczba` służącą do przechowywania liczb całkowitych. Klasa powinna udostępniać następujące metody publiczne:
- `wczytaj` wczytującą wartość liczby ze standardowego wejścia,
- `wypisz` wypisującą wartość liczby na standardowe wyjście,
- `nadaj_w` nadającą przechowywanej liczbie wartość podaną w argumencie metody,
- `wartosc` zwracającą wartość przechowywanej liczby,
- `abs` zwracającą wartość bezwzględną przechowywanej liczby.

Napisz klasę `liczba` w taki sposób, żeby dostęp do zawartych w niej danych był możliwy tylko za pośrednictwem metod tej klasy.

1.16 Napisz klasę `portfel`, przechowującą kwotę posiadanych pieniędzy. Klasa `portfel` powinna udostępniać następujące publiczne metody:
- `inicjuj` nadającą przechowywanej kwocie wartość 0.
- `zarobki` dodającą do przechowywanej kwoty wartość podaną w argumencie.
- `wydatki` odejmującą od przechowywanej kwoty wartość podaną w argumencie.
- `zawartosc` zwracającą jako wartość przechowywaną kwotę.

Napisz klasę `portfel` w taki sposób, żeby dostęp do zawartych w niej danych był możliwy tylko za pośrednictwem metod tej klasy.

1.17 **(r)** Napisz klasę `punkt` służącą do przechowywania współrzędnych punktu w dwuwymiarowym kartejzańskim układzie współrzędnych. Napisz metody do wczytywania i wypisywania współrzędnych. Zadeklaruj wszystkie pola klasy jako prywatne.

1.18 **(r)** Napisz klasę `punkt3` służącą do przechowywania współrzędnych punktu w trójwymiarowym kartejzańskim układzie współrzędnych. Napisz metody do wczytywania i wypisywania współrzędnych. Zadeklaruj wszystkie pola klasy jako prywatne.

1.19 **(r)** Napisz funkcję `rzutuj`, która otrzymuje jako argument obiekt typu `punkt3` z zadania 1.18 i zwraca jako wartość obiekt typu `punkt` z zadania 1.17 będący prostopadłym rzutem punktu otrzymanego w argumencie na płaszczyznę wyznaczoną przez dwie pierwsze współrzędne

1.20 **(r)** Do klasy `punkt` z zadania 1.17 dopisz metodę `rzutuj`, która otrzy-

muje jako argument obiekt typu `punkt3` z zadania 1.18 i przypisuje polom obiektu, na rzecz którego została wywołana, współrzędne prostopadłego rzutu punktu otrzymanego w argumencie na płaszczyznę wyznaczoną przez dwie pierwsze współrzędne.

1.21 Napisz klasę `zespolona` służącą do przechowywania liczb zespolonych. Udostępnij dostęp do pól tej klasy wyłącznie za pomocą publicznych metod.

1.22 Napisz funkcję `suma`, która dostaje jako argumenty dwa obiekty klasy `zespolone` z zadania 1.21 i zwraca jako wartość ich sumę.

1.23 Napisz klasę `dane_os` służącą do przechowywania danych osobowych. Klasa `dane_os` powinna posiadać prywatne pola `imie`, `nazwisko` i `adres` dostępne wyłącznie za pośrednictwem publicznych metod.

1.24 Do klasy `dane_os` z zadania 1.23 dopisz metodę `wypisz` wypisującą przechowywane dane osobowe.

1.25 Napisz klasę `tablica`, służącą do przechowywania 10-elementowej tablicy. Dostęp do poszczególnych elementów tablicy powinien być wyłącznie za pomocą publicznej metody `at`, która dla podanego indeksu zwraca referencję do odpowiedniego elementu tablicy. W przypadku podania indeksu spoza zakresu od 0 do 9 metoda `at` powinna zwrócić referencję do pierwszego elementu tablicy.

1.26 **(r)** Napisz klasę `ukryta_liczba`, która przechowuje liczbę całkowitą w prywatnym polu `liczba` i udostępnia publiczną metodę `zeruj`, przypisującą wartość `0` polu `liczba`.

1.27 **(r)** Napisz funkcję `inkrementuj`, która zwiększa o jeden wartość pola `liczba` obiektu typu `ukryta_liczba` z zadania 1.26, do którego referencję funkcja dostała w argumencie.

1.28 **(r)** Do klasy `ukryta_liczba` z zadania 1.26 dopisz metodę `inkrementuj`, która zwiększa o jeden wartość pola `liczba` otrzymanego w argumencie obiektu typu `ukryta_liczba`

1.29 Napisz klasę `wektor` służącą do przechowywania dziesięciowymiarowych wektorów. Klasa `wektor` powinna udostępniać następujące publiczne metody:

- `wypisz` wypisującą wartość wektora na standardowym wyjściu,
- `wczytaj` wczytującą wartość wektora ze standardowego wejścia,
- `dodaj` dodającą do przechowywanego wektora wektor otrzymany w argumencie.

Wszystkie pola klasy `wektor` powinny zostać zadeklarowane jako prywatne.

1.30 Napisz funkcję `porownaj`, która dostaje w argumentach dwa obiekty typu `wektor` z zadania 1.29 i zwraca jako wartość `true` jeżeli pierwszy z otrzymanych w argumentach wektorów jest dłuższy oraz `false` w przeciwnym wypadku.

1.31 Napisz klasę `odcinek` przechowująca współrzędne w dwuwymiarowym kartezjańskim układzie współrzędnych początku i końca odcinka. Klasa `odcinek` powinna udostępniać następujące publiczne metody:
- `wczytaj` wczytującą współrzędne początku i końca odcinka ze standardowego wejścia,
- `wypisz` wypisującą współrzędne początku i końca odcinka na standardowym wyjściu,
- `przeciecie` o argumencie typu `odcinek` zwracającą `true`, jeżeli otrzymany w argumencie odcinek przecina się z tym, którego dane są przechowywane w obiekcie.

1.32 Napisz funkcję zwracającą jako wartość długość odcinka, którego dane przechowywane są w otrzymanym w argumencie obiekcie typu `odcinek` z zadania 1.31.

1.33 **(r)** Napisz klasę `wskaznik` zawierającą jedno pole prywatne `wsk` typu wskaźnik do zmiennej typu `int`. Klasa `wskaznik` powinna udostępniać następujące publiczne metody:
- `utworz`, która dla otrzymanej w argumencie dodatniej liczby całkowitej `n`, rezerwuje pamięć dla `n`-elementowej tablicy o elementach typu `int` i zapisuje w polu `wsk` wskaźnik do nowo utworzonej tablicy,
- `zwroc` zwracającą jako wartość wskaźnik przechowywany w polu `wsk`,
- `zwolnij` zwalniającą obszar pamięci wskazywany przez pole `wsk` i nadająca temu polu wartość `NULL`,
- `kopiuj`, która otrzymuje jako argument referencję `ref` do zmiennej typu `wskaznik` i dokonuje przypisania `ref.wsk=wsk`.

1.34 **(r)** Napisz funkcję `przepisz`, która dostaje jako argumenty wskaźnik `t` do tablicy o elementach typu `int` oraz referencję `ref` do zmiennej typu `wskaznik` z zadania 1.33 i przypisuje wskaźnik `t` do pola `wsk` obiektu, do którego referencję przechowuje zmienna `ref` .

1.35 **(r)** Do klasy `wskaznik` z zadania 1.33 dopisz:
- bezargumentowy konstruktor, przypisujący do pola `wsk` wartość `NULL`,
- destruktor zwalniający obszar pamięci wskazywany przez pole `wsk` (o ile ma ono wartość różną od `NULL`).

1.36 **(r)** Napisz klasę `identyfikator`, która udostępnia tylko jedną publiczną bezargumentową metodę `id`. Metoda `id` powinna zwracać, za każdym razem inną, nieujemną liczbę całkowitą.

1.37 **(r)** Napisz klasę `identyfikator2`, która udostępnia tylko jedną publiczną bezargumentową metodę `id`. Metod `id` powinna zwracać kolejne liczby całkowite począwszy od 0.

1.38 **(r)** Napisz klasę `semafor_bin`, której obiekty w każdym momencie są w jednym z dwóch stanów *wolny* lub *zajęty*. Bezpośrednio po utwo-

rzeniu obiekt typu `semafor_bin` powinien być w stanie *wolny*. Klasa `semafor_bin` powinna udostępniać następujące publiczne metody:
- `rezerwuj`, której wywołanie zmienia stan semafora z *wolny* na *zajęty* (w przypadku, gdy semafor jest już w stanie *zajęty*, metoda nie powinna zmieniać jego stanu),
- `zwolnij`, której wywołanie zmienia stan semafora z *zajęty* na *wolny* (w przypadku, gdy semafor jest już w stanie *wolny*, metoda nie powinna zmieniać jego stanu),
- `stan` zwracającą wartość `true` gdy semafor jest w stanie *wolny* i `false` w przeciwnym wypadku.

Wszystkie pola klasy `semafor_bin` powinny być prywatne.

1.39 Napisz klasę `semafor`, której obiekty mogą przyjmować stany ze zbioru $\{0, 1, \ldots, n-1\}$, gdzie n jest argumentem konstruktora. Bezpośrednio po utworzeniu obiekt typu `semafor` powinien być w stanie 0. Klasa `semafor` powinna posiadać następujące publiczne metody:
- `semafor(unsigned int n)` konstruktor, inicjujący obiekt, który może przyjmować stany ze zbioru $\{0, 1, \ldots, n-1\}$,
- bezargumentowy konstruktor, inicjujący obiekt, który może przyjmować stany ze zbioru $\{0, 1\}$,
- `rezerwuj`, której wywołanie zwiększa wartość stanu semafora o jeden (w przypadku, gdy semafor jest w stanie o maksymalnej wartości metoda nie powinna niczego robić),
- `zwolnij`, której wywołanie zmniejsza wartość stanu semafora o jeden (w przypadku, gdy semafor jest w stanie 0, metoda nie powinna niczego robić),
- `stan` zwracającą wartość `true` gdy semafor nie osiągnął jeszcze maksymalnej wartości stanu i `false` w przeciwnym wypadku.

Wszystkie pola klasy `semafor` powinny być prywatne.

1.40 Popraw klasę `semafor` z zadania 1.39 w taki sposób, żeby pamiętała identyfikatory procesów rezerwujących zasoby przy pomocy obiektów tej klasy. W tym celu metody `rezerwuj` i `zwolnij` powinny mieć całkowitoliczbowy argument `id`, w którym będą otrzymywać identyfikator procesu rezerwującego/zwalniającego zasób. Poprawiona klasa powinna ponadto posiadać publiczną metodę `wypisz` wypisującą na standardowym wyjściu identyfikatory procesów, które w danym momencie posiadają rezerwację zasobu.

1.41 Napisz klasę `macierz`, służącą do przechowywania macierzy kwadratowych liczb wymiernych. Klasa `macierz` powinna zawierać:
- publiczne pole `tab`, zawierające wskaźnik do macierzy,
- publiczne pole `n`, zawierające rozmiar macierzy,
- konstruktor, kóry dostaje w argumencie dodatnią liczbę całkowitą n i tworzy macierz o wymiarach $n \times n$,

— destruktor, który zwalnia pamięć zarezerwowaną przez obiekt.

1.42 Napisz klasę `staly_napis`, służącą do przechowywania napisów. Obiekt klasy `staly_napis` powinien od powstania do usunięcia przechowywać ten sam napis otrzymany w konstruktorze. Klasa `staly_napis` powinna udostępniać:
— konstruktor otrzymujący jako argument napis, który ma być przechowywany w obiekcie,
— metodę `at`, która zwraca wartość znaku znajdującego się w napisie pod indeksem podanym w argumencie.
Wszystkie pola w klasie `staly_napis` powinny być prywatne.

1.43 **(r)** Napisz klasę `osoba` o dwóch polach prywatnych `imie` i `nazwisko`. Klasa `osoba` powinna udostępniać następujące publiczne metody:
— konstruktor otrzymujący jako argumenty imię i nazwisko, które mają być przechowywane w obiekcie,
— `wczytaj` wczytującą ze standardowego wejścia wartości do pól obiektu,
— `wypisz` wypisującą na standardowym wyjściu wartości pól przechowywanych w obiekcie.

1.44 **(r,!)** Napisz funkcję, która dostaje jako argument całkowitą liczbę dodatnią `n` i zwraca jako wartość wektor `n` obiektów typu `osoba` z zadania 1.43. Elementy zwracanego wektora powinny przechowywać imię i nazwisko `Jan Kowalski`.

1.45 **(r,!,*)** Napisz klasę `tab_osob` przechowującą tablicę `tab` wskaźników do obiektów klasy `osoba` z zadania 1.43. Klasa `tab_osob` powinna udostępniać:
— konstruktor, który dostaje jako argumenty nieujemną liczbę całkowitą `n` oraz napisy `imie` i `nazwisko`, tworzy `n` obiektów klasy `osoba` zainicjowanych wartościami zmiennych `imie` i `nazwisko`, tworzy `n`-elementową tablicę `tab` oraz inicjuje jej komórkami wskaźnikami do nowo utworzonych obiektów klasy `osoba`,
— metodę `at`, która otrzymuje jako argument nieujemną liczbą całkowitą `ind` i zwraca jako wartość referencję do obiektu wskazywanego przez element tablicy `tab` o indeksie `ind`,
— destruktor zwalniający pamięć zarezerwowaną przez obiekt.

1.46 **(r,*)** Napisz klasę `kolejka` będącą implementacją klasycznej kolejki przechowującej liczby całkowite. Klasa `kolejka` powinna udostępniać następujące publiczne metody:
— bezargumentowy konstruktor tworzący pustą kolejkę,
— konstruktor kopiujący,
— destruktor zwalniający pamięć zaalokowaną przez obiekt,
— `pierwszy` zwracającą jako swoją wartość pierwszy element kolejki,
— `usun_pierwszy` usuwającą pierwszy element kolejki,

— `dodaj_na_koniec`, dodającą na koniec kolejki liczbę całkowitą otrzymaną w argumencie,
— `pusta` zwracającą `true` jeżeli kolejka jest pusta i `false` w przeciwnym wypadku.

1.47 (*) Napisz klasę `stos` będącą implementacją klasycznego stosu przechowującego liczby całkowite. Klasa `stos` powinna udostępniać następujące publiczne metody:
— bezargumentowy konstruktor tworzący pusty stos,
— konstruktor kopiujący,
— destruktor zwalniający pamięć zaalokowaną przez obiekt,
— `z_wierzchu` zwracająca go jako swoją wartość element z wierzchu stosu,
— `usun_z_wierzchu` usuwający element znajdujący się na wierzchu stosu,
— `poloz_na_stos` kładący na wierzchu stosu liczbę całkowitą otrzymaną w argumencie,
— `pusty` zwracającą `true` jeżeli stos jest pusty i `false` w przeciwnym wypadku.

1.48 Dopisz konstruktor kopiujący do klasy `macierz` z zadania 1.41.

1.49 Napisz funkcję, która dostaje jako argumenty dwa obiekty typu `macierz` z zadania 1.41 i zwraca jako wartość kopię tego z nich, który ma mniej komórek równych 0. Jeżeli w macierzach przechowywanych w otrzymanych w argmentach obiektach jest tyle samo zer, funkcja powinna zwrócić jako wartość kopię pierwszego argumentu.

1.50 Zmień funkcję z zadania 1.49 w taki sposób, żeby w argumentach zamiast dwóch obiektów typu `macierz` dostawała stałe referencje do nich.

1.51 Napisz klasę `wektorn` służącą do przechowywania wektorów w przestrzeniach wielowymiarowych. Wszystkie pola w klasie `wektorn` powinny być prywatne. Wektor powinien udostępniać następujące metody publiczne:
— konstruktor o jednym całkowitoliczbowym argumencie `n` tworzący `n`-wymiarowy wektor,
— destruktor zwalniający pamięć zaalokowaną przez obiekt,
— `at` zwracający referencję do współrzędnej wektora o indeksie podanym w argumencie metody,
— `wymiar` zwracającą liczbę wymiarów wektora przechowywanego w obiekcie.

1.52 Napisz funkcję, która dostaje dwa argumenty typu `wektorn` z zadania 1.51 i zwraca jako wartość sumę otrzymanych w argumentach wektorów. Jeżeli wektory mają różną liczbę wymiarów, funkcja powinna zwrócić wektor równy temu z otrzymanych w argumentach wektorów, który ma większą liczbę wymiarów.

1.53 Zmień funkcję z zadania 1.52 w taki sposób, żeby w argumentach zamiast dwóch obiektów typu `wektorn` dostawała stałe referencje do nich.

1.54 Zaimplementuj klasę `napis` przechowującą napis w prywatnej tablicy znaków. Klasa ta powinna mieć następujące metody publiczne:
- bezparametrowy konstruktor tworzący pusty napis,
- konstruktor kopiujący,
- konstruktor, którego parametrem jest napis przechowywany w tradycyjny sposób, czyli w tablicy o elementach typu `char`, w której koniec napisu jest zaznaczony przez znak o numerze 0,
- destruktor usuwający wszystkie dynamiczne struktury danych przechowywane przez obiekt,
- metodę `dopisz` o jednym parametrze która do istniejącego napisu „dokleja" na końcu napis podany w parametrze (utwórz dwie wersje metody `dopisz` - z parametrem typu `napis` i tablicą znaków),
- metodę `dlug`, zwracającą długość przechowywanego napisu.

1.55 **(*)** Zaimplementuj klasę `tablica` służącą do przechowywania liczb całkowitych. Klasa `tablica` powinna udostępniać:
- bezparametrowy konstruktor tworzący pustą tablicę,
- konstruktor kopiujący,
- destruktor usuwający wszystkie dynamiczne struktury danych przechowywane przez obiekt,
- metodę `wartosc` zwracającą wartość komórki tablicy o indeksie podanym w argumencie tej metody,
- metodę `przypisz` nadającą komórce tabeli o podanym w pierwszym argumencie indeksie wartość podaną w drugim argumencie.

Obiekt tej klasy powinien zachowywać się jak tablica, która „rośnie" w miarę potrzeb. W przypadku użycia metody `przypisz` z indeksem i ($i > 0$) spoza zakresu dozwolonych indeksów tablica powinna być automatycznie powiększona do tablicy $i + 1$ elementowej.

1.56 **(*)** Napisz klasę `lista` służącą do przechowywania listy zakupów. Poszczególne pozycje listy mają się składać z dwóch elementów: nazwy towaru i ilości w jakiej planujemy go zakupić. Lista powinna udostępniać następujące publiczne metody:
- bezargumentowy konstruktor tworzący pustą listę,
- konstruktor kopiujący,
- destruktor,
- `dodaj` otrzymującą w argumentach nazwę towaru oraz ilość tego towaru jaką chcemy zakupić i dodającą te informacje do przechowywanej listy zakupów,
- `wypisz` wypisującą na standardowym wyjściu wszystkie elementy listy,
- `usun` usuwającą z listy towar podany w argumencie,

— `wyczysc` usuwającą wszystkie elementy listy.

1.2. Dziedziczenie

1.57 **(r)** Napisz klasę `figura` posiadająca publiczne pola obwód i pole. Napisz klasy pochodne klasy figura służące do przechowywania danych różnych konkretnych figur. Klasy pochodne powinny posiadać publiczne pola służące do przechowywania ich wymiarów (różne w zależności od rodzaju przechowywanych figur).

1.58 **(r,!)** W zadaniu 1.57 zamiast klas zdefiniuj analogiczne struktury.

1.59 Napisz klasę `ubranie` posiadającą publiczne pola `material` i `kolor`. Napisz klasy `spodnie`, `koszula` i `czapka` pochodne klasy `ubranie`. Klasy pochodne powinny posiadać następujące pola publiczne:
— klasa `spodnie` pola `dlugosc` i `w_pasie`,
— klasa `koszula` pola `dlugosc` i `w_klatce`,
— klasa `czapka` pole `obwod`.

1.60 Napisz klasę `mebel` zawierającą publiczne pola `producent` i `kolekcja` oraz klasy `krzeslo`, `stol`, `szafka` pochodne klasy `mebel`. Klasy pochodne powinny posiadac następujące pola publiczne:
— klasa `krzeslo` pole `obicie`,
— klasa `stol` pola `szerokosc` i `dlugosc`,
— klasa `szafka` pola `szerokosc`, `wysokosc` i `glebokosc`.

1.61 Napisz klasę `zwierze` udostępniającą publiczne pola `gatunek` oraz `imie`. Napisz klasy `zmija`, `rys`, `orzel` pochodne klasy `zwierze`, służące do przechowywania informacji o zwierzętach konkretnych gatunków.
— klasa `zmija` powinna posiadać publiczne pole `dlugosc`,
— klasa `rys` powinna posiadać publiczne pole `dlugosc` i `wysokosc`,
— klasa `orzel` powinna posiadać publiczne pole `dlugosc` i `rozpietosc_skrzydel`

Konstruktory klas pochodnych powinny nadawać polu `gatunek` odpowiednią wartość.

1.62 **(r)** Napisz klasę `atrakcja` posiadającą chronione pola `cena`, `nazwa` i `opis` oraz publiczne metody zwracające wartości tych pól w taki sposób, by nie można było ich modyfikować z zewnątrz klasy. Napisz klasy `kolejka`, `zamek`, `film` pochodne klasy `atrakcja`. Klasy pochodne powinny posiadać następujące pola prywatne
— `kolejka` pola `godz_odjazdu` i `godz_przyjazdu`,
— `zamek` pole `czas_zwiedzania`,
— `film` pola `czas_trwania` i `tytul`

Napisz w klasach pochodnych metody zwracające wartości ich nowych pól w taki sposób, żeby nie można było ich zmieniać. Zdefiniuj w klasach

pochodnych metodę `inicjuj`, nadającą wszystkim polom klasy wartości podane w argumentach metody.

1.63 **(r)** Napisz klasę `lista` służącą do przechowywania listy liczb całkowitych. Klasa `lista` powinna udostępniać następujące metody publiczne:
- bezargumentowy konstruktor tworzący pustą listę,
- konstruktor kopiujący,
- `dodaj_przod` dodającą na początek listy liczbę całkowitą podaną w argumencie,
- `dodaj_tyl` dodającą na koniec listy liczbę całkowitą podaną w argumencie,
- `usun_przod` usuwającą pierwszy element listy,
- `usun_tyl` usuwającą ostatni element listy,
- `pierwszy_el` zwracającą wartość pierwszego elementu listy,
- `ostatni_el` usuwającą wartość ostatniego elementu listy,
- `pusta_lista` zwracającą `true` jeżeli lista nie zawiera żadnego elementu oraz `false` w przeciwnym wypadku.

1.64 **(r)** Napisz klasę `kolejka` z zadania 1.46 wykorzystującą do przechowywania danych prywatne pole typu `lista` z zadania 1.63.

1.65 **(r)** Napisz klasę `kolejka` z zadania 1.46 jako klasę pochodną klasy `lista` z zadania 1.63.

1.66 Napisz klasę `stos` z zadania 1.47 wykorzystującą do przechowywania danych prywatne pole typu `lista` z zadania 1.63.

1.67 Napisz klasę `stos` z zadania 1.47 jako klasę pochodną klasy `lista` z zadania 1.63.

1.68 **(r,!,*)** Napisz klasę `lepsza_lista` pochodną klasy `lista` z zadania 1.63. Wewnątrz klasy `lepsza_lista` powinna zostać zaimplementowana klasa `iterator`, której obiekty mają „wskazywać" na pojedyncze elementy listy. Klasa `iterator` powinna udostępniać następujące publiczne metody:
- konstruktor domyślny inicjujący iterator wskazujący na pierwszy element listy podanej w argumencie konstruktora,
- `element`, zwracającą referencję do wskazywanego elementu listy (liczby całkowitej),
- `nastepny` przesuwającą iterator o jedną pozycję do przodu (to znaczy, że po wywołaniu tej metody obiekt typu `iterator` będzie wskazywał następny w kolejności element listy),
- `poprzedni` przesuwającą iterator o jedną pozycję do tyłu (to znaczy, że po wywołaniu tej metody obiekt typu `iterator` będzie wskazywał poprzedni w kolejności element listy).
- `poczatek` zwracającą `true` jeżeli `iterator` wskazuje na pierwszy element listy i `false` w przeciwnym wypadku,

— `koniec` zwracającą `true` jeżeli `iterator` wskazuje na ostatni element listy i `false` w przeciwnym wypadku,

1.69 **(r)** Funkcję `zeruj`, która otrzymuje jako argument referencję do obiektu klasy `lepsza_lista` i nadaje wartość `0` wszystkim elementom otrzymanej listy.

1.70 **(r)** Napisz klasę `stala1` posiadającą stałe publiczne pole `i` typu `int` o wartości `5`.

1.71 **(r)** Napisz klasę `stala2` posiadającą stałe publiczne pole `d` typu `double`. Wartość pola `d` powinna być podawana przy tworzeniu obiektu klasy `stala2` jako argument konstruktora.

1.72 **(r)** Napisz klasę `stale` pochodną typu `stala2` z zadania 1.71. Klasa `stale` powinna posiadać publiczne pole `liczba` typu `stala2`. Zarówno dziedziczone pole `d` jak i publiczne pole `liczba` powinny być zainicjowane wartościami liczbowymi podanymi jako argumenty konstruktora klasy `stale`.

1.73 Napisz klasę `l_stala` zawierającą publiczne pole `liczba` typu `const unsigned int`. Napisz konstruktor klasy `l_stale`, który dostaje jako argument dodatnią liczbę całkowitą n i nadaje polu liczba losową wartość z zakresu od 1 do n.

1.74 Napisz klasę `l_stala2` zawierająca pole `liczba1` typu `l_stala` z zadania 1.73 i pole `liczba2` typu `const unsigned int`. Napisz konstruktor klasy `l_stala2`, który otrzymuje jako argument dodatnią liczbę całkowitą n, po którego wykonaniu pola `liczba1` i `liczba2` będą przechowywać tą samą losową wartość z zakresu od 1 do n.

1.75 Napisz klasę `superwektor` pochodną klasy `wektorn` z zadania 1.51, która posiada dodatkowo następujące publiczne metody:
— bezargumentowy konstruktor tworzący wektor dwuwymiarowy,
— `dlugosc` zwracającą długość przechowywanego wektora.

1.76 Napisz klasę `lwektor` dziedziczącą po klasie `l_stala` z zadania 1.73 i zawierającą publiczne pole `wek` typu `wektorn` z zadania 1.51. Klasa `lwektor` powinna posiadać konstruktor, który otrzymuje jako wartość dodatnią liczbę całkowitą n, inicjuje wartość dziedziczonego pola `liczba` losową wartością m z zakresu 1 do n i inicjuje pole `wek` jako wektor o m wymiarach.

1.77 Napisz klasę `stala_figura` służącą do przechowywania danych figur geometrycznych. Klasa ta powinna udostępniać następujące publiczne metody:
— konstruktor, który jako argumenty otrzymuje pole, obwód i rodzaj przechowywanej figury,
— `pole`, zwracającą pole przechowywanej figury,
— `obwod` zwracającą obwód przechowywanej figury,

— `rodzaj` wypisującą na standardowym wyjściu rodzaj przechowywanej figury (np. kwadrat, trójkąt etc.)

Napisz klasy pochodne klasy `stala_figur` służące do przechowywania danych różnych konkretnych rodzajów figur geometrycznych (np. kwadratów, trójkątów). Klasy pochodne powinny posiadać konstruktory, których parametrami są wymiary przechowywanych figur (różne w zależności od rodzaju figury) oraz publiczne metody udostępniające poszczególne wymiary figur. Klasy powinny być napisane w taki sposób, żeby metody `pole` i `obwód` wyświetlały wartości obliczone na podstawie podanych w konstruktorach wymiarów.

1.78 Napisz klasę `lepszy_int` zawierającą publiczne pole `liczba` typu `int`. Dla klasy `lepszy_int` zdefiniuj konstruktor, który nadaje polu `liczba` wartość podaną w argumencie. Nie definiuj dla klasy `lepszy_int` konstruktora bezargumentowego.

1.79 Napisz klasę `para` przechowującą dwa pola typu `lepszy_int` z zadania 1.78. Napisz konstruktor klasy `para` nadający obu polom obiektu wartości liczb całkowitych podanych w argumentach.

1.80 **(r)** Napisz klasę `liczba` nie zawierającą, żadnego pola. Zdefiniuj klasy `calkowita` i `wymierna` dziedziczące publicznie po klasie `liczba` zawierające dodatkowo pole `wartosc` typu odpowiednio `int` i `double`.

1.81 **(r)** Napisz funkcję `kopiuj`, która dostaje jako argument tablicę elementów będących wskaźnikami do typu `liczba` z zadania 1.80 i jego typów pochodnych oraz jej rozmiar, tworzy kopię otrzymanej w argumencie tablicy i zwraca ją jako swoją wartość.

1.82 Napisz klasę `list_figur` przechowującą listę klas pochodnych klasy `figura` z zadania 1.57. Klasa `lista_figur` powinna udostępniać następujące publiczne metody:

— konstruktor otrzymujący jako argument maksymalną liczbę elementów listy,

— `dodaj` dodająca na koniec listy obiekt, do którego wskaźnik metoda otrzymała w argumencie,

— `ostatni` zwracający jako wartość wskaźnik do ostatniego spośród przechowywanych w liście obiektów,

— `usun` usuwająca ostatni element listy,

— `srednia`, zwracająca jako wartość średnią z obwodów przechowywanych na liście figur.

Rozdział 2

Polimorfizm

2.1 **(r)** Napisz klasę `bazowa` oraz jej klasy pochodne `pochodna1` i `pochodna2`. Powyższe trzy klasy powinny udostępniać następujące metody publiczne:
- `typ_wskaznika` wypisująca na standardowym wejściu typ wskaźnika, przy pomocy którego wywołana została ta metoda,
- `typ_obiektu` wypisująca na standardowym wyjściu typ obiektu wskazywanego przez wskaźnik, przy pomocy którego wywołana została ta metod.

2.2 **(r)** Napisz klasę `liczba` służącą do przechowywania liczb wymiernych. Klasa `liczba` powinna posiadać publiczne pole `re` typu `double` oraz następujące metody publiczne:
- wirtualną metodę `modul` zwracającą moduł przechowywanej liczby,
- `wieksza` otrzymującą w argumencie referencję `ref` do obiektu klasy `liczba` i zwracający jako wartość `true` jeżeli moduł liczby `ref` jest większy od modułu liczby przechowywanej w obiekcie, na rzecz którego wywoływana jest metoda oraz `false` w przeciwnym wypadku.

2.3 **(r)** Napisz klasę `zespolone` pochodną klasy `liczba` z zadania 2.2 posiadającą dodatkowo publiczne pole `im` typu `int`. Przeciąż w klasie `zespolone` metodę `modul`. Czy metodę `wiekszy` też trzeba przeciążyć?

2.4 **(r)** Zaimplementuj klasę `funkcja` posiadającą publiczne pole `x` oraz czysto wirtualną metodę `wartosc`, która w klasach pochodnych będzie zwracała wartość funkcji przechowywanej w obiekcie w punkcie `x`.

2.5 **(r)** Zaimplementuj klasę `funkcja_liniowa` pochodną klasy `funkcja` z zadania 2.4. Klasa `funkcja_liniowa` powinna zawierać publiczne pola `a` i `b` oraz przeciążoną metodę `wartosc` w taki sposób, żeby zwracała wartość funkcji `a*x+b`.

2.6 **(r)** Napisz funkcję `bisekcja`, która otrzymuje jako argumenty wskaźnik do obiektu klasy pochodnej klasy `funkcja` z zadania 2.4, liczby `p`, `k` oraz `d` i szuka miejsca zerowego przekazanej w argumencie funkcji metodą bisekcji w przedziale od `p` do `k`. Funkcja ma zwrócić miejsce zerowe z dokładnością do `d`. Jeżeli wartości funkcji na końcach zadanego przedziału są tego samego znaku to funkcja może zwrócić cokolwiek.

2.7 Napisz klasę `liczba` posiadającą publiczne czysto wirtualne metody `wczytaj` i `wypisz`. Napisz klasy `nint` i `ndouble` dziedziczące publicznie po klasie `liczba` i posiadające publiczne pola `wartosc` odpowiednio typu `int` i `double`. Przeciąż dla klas `nint` i `ndouble` metody `wczytaj` i `wypisz` odpowiednio wczytującą ze standardowego wejścia i wypisującą na standardowym wyjściu zawartość pola `wartosc`.

2.8 Napisz funkcję `wypisz_tab` otrzymującą jako argument tablicę o elementach typu `liczba*` z zadania 2.7 oraz jej rozmiar i wypisującą war-

tości przechowywane przez obiekty wskazywane przez elementy otrzymanej w argumencie tablicy.

2.9 Napisz program, który wczytuje ze standardowego wejścia pewną, ustaloną przez użytkownika, liczbę wartości typów `int` i `double`, zapamiętuje je w obiektach typów `ntint` i `ndouble` z zadania 2.7 i wypisuje przy pomocy funkcji `wypisz_tab` z zadania 2.8

2.10 Napisz klasę `towar` posiadającą publiczne pola `nazwa`, `cena` oraz `ilosc` i wirtualną metodę `opis` wyświetlającą na standardowym wyjściu wszystkie informacje przechowywane w obiekcie.

2.11 Napisz następujące klasy pochodne klasy `towar` z zadania 2.10:

a) `gwozdzie` posiadające dodatkowe publiczne pola `długosc`, `grubosc` i `rodzaj_lepka`,

b) `papier_scierny` posiadające dodatkowe publiczne pola `ziarnistosc` i `szerokosc`,

c) `meble` posiadające dodatkowe pole `kolekcja`.

Wszystkie klasy pochodne klasy `towar` powinny mieć metodę `opis` przeciążoną w taki sposób, żeby wykorzystać oryginalny kod tej metody.

2.12 Zaimplementuj klasę `szafa` pochodną klasy `meble` z zadania 2.11. Klasa `szafa` powinna posiadać publiczne pola `wysokosc`, `szerokosc` i `glebokosc`. Metoda `opis` powinna zostać przeciążona w taki sposób, żeby wykorzystać kod metody `opis` z klas bazowych.

2.13 Napisz funkcję `wypisz`, która dostaje jako argument wektor `vec` wskaźników do obiektów klasy `towar` z zadania 2.10 i wypisuje przy pomocy metody `opis` opisy wszystkich obiektów, do których wskaźniki przechowywane są w wektorze `vec`.

2.14 Napisz program, który wczytuje ze standardowego wejścia dane różnych towarów, przechowuje je w obiektach klas pochodnych klasy `towar` zdefiniowanych w zadaniach 2.11 oraz 2.12 i na koniec wypisuje opisy wczytanych towarów za pomocą funkcji `wypisz`.

2.15 Zaimplementuj klasę `czworokat` posiadającą pola chronione `a`, `b`, `c` i `d` służące do przechowywania długości boków czworokąta. Klasa `czworokat` powinna posiadać:

— metodę `wypisz` wypisującą na standardowym wyjściu długości wszystkich czterech boków,

— czysto wirtualną metodę `pole` zwracającą jako wartość pole czworokąta,

— czteroargumentowy konstruktor nadający polom `a`, `b`, `c` i `d` wartości otrzymane w argumentach.

2.16 Zaimplementuj następujące klasy pochodne klasy `czworokat` z zadania 2.15:

— `prostokat` posiadającą następujące publiczne metody:

  — dwuargumentowy konstruktor nadający polom `a` i `c` wartość otrzy-

maną w pierwszym argumencie,zaś polom `b` i `d` wartość otrzymaną w drugim argumencie,
- `wymiary`, która otrzymuje dwa argumenty `w1` oraz `w2` i nadaje polom `a` i `c` wartość `w1`, zaś polom `b` i `d` wartość `w2`.

— `kwadrat` posiadającą następujące publiczne metody:
- jednoargumentowy konstruktor nadający polom `a`, `b`, `c` i `d` wartość otrzymaną w argumencie.
- `wymiar` nadającą polom `a`, `b`, `c` i `d` jedną wartość otrzymaną w argumencie.

Klasy `prostokat` i `kwadrat` powinny mieć odpowiednio przeciążoną metodę `pole`.

2.17 Napisz funkcję `wypisz_pola`, która otrzymuje jako argument tablicę o elementach typu `czworokat *`, gdzie `czworokat` jest typem zdefiniowanym w zadaniu 2.15 oraz jej rozmiar i wypisuje pola wszystkich przechowywanych w tablicy czworokątów.

2.18 Napisz program, który wczytuje ze standardowego wejścia wymiary kwadratów i prostokątów, przechowuje ich wymiar wykorzystując obiekty klas `prostokat` i `kwadrat` z zadania 2.16 i wypisuje ich pola przy wykorzystaniu funkcji `wypisz_pola` z zadania 2.17.

2.19 Napisz klasę `wierzcholek` posiadającą stałe publiczne pole `wartosc` typu `int` oraz prywatne pola `ojciec`, `lewy_syn`, `prawy_syn` będące wskaźnikami na typ `wierzcholek`

Napisz konstruktor klasy `wierzcholek`, który nadaje polu `wartosc` wartość otrzymaną jako argument, zaś pozostałym polom przypisuje wartość `NULL`.

2.20 **(*)** Napisz klasę abstrakcyjną `drzewo` zawierającą następujące metody publiczne:
- konstruktor, który dla podanej w argumencie liczby całkowitej `wartosc` tworzy drzewo, którego jedyny wierzchołek przechowuje liczbę `wartosc`
- destruktor zwalniający zaalokowaną przez obiekt pamięć,
- metodę `wstaw`, która otrzymuje trzy argumenty: liczbę cakowitą `wartosc`, wskaźnik `wsk` do obiektu typu `wierzcholek` z zadania 2.19 i wartość logiczną `strona`, tworzy nowy obiekt typu `wierzcholek` i dodaje go do drzewa jako syna wierzchołka wskazywanego przez `wsk`. Jeżeli zmienna `strona` jest równa `true` to nowy wierzchołek powinien zostać dodany jako lewy syn `wsk` (jeżeli `wsk` ma już lewego syna, to metoda `wstaw` nie powinna dodawać nowego wierzchołka). Analogicznie, jeżeli `strona` jest równa `false` i `wsk` nie ma prawego syna, to metoda `wstaw` powinna dodać nowy wierzchołek do drzewa jako prawego syna `wsk`.
- czysto wirtualną metodę `nastepny`, która dla otrzymanego w argu-

mencie wskaźnika do typu `wierzcholek` z zadania 2.19 wskazującego na pewien wierzchołek drzewa zwraca wskaźnik do kolejnego według pewnej kolejności wierzchołka drzewa. Dla ostatniego wierzchołka metoda powinna zwrócić pierwszy wierzchołek według tej samej kolejności.

2.21 **(*)** Napisz klasy `drzewo_inorder`, `drzewo_preorder` `drzewo_postorder` pochodne klasy `drzewo` z zadania 2.19, w których metoda `nastepny` zwraca kolejny wierzchołek odpowiednio w kolejności inorder, preorder i postorder.

2.22 **(r,!)** Napisz abstrakcyjną klasę `kolejka` definiującą interfejs kolejki liczb całkowitych. Klasa `kolejka` powinna posiadać następujące publiczne czysto wirtualne metody:
- `pierwszy` zwracającą wartość pierwszego elementu kolejki,
- `usun-pierwszy` usuwającą pierwszy element kolejki
- `dodaj_na_koniec`, dodającą na koniec kolejki liczbę całkowitą otrzymaną w argumencie,
- `pusta` zwracającą `true`, jeżeli kolejka jest pusta i `false` w przeciwnym wypadku.

Klasa powinna udostępniać także wirtualny destruktor.

2.23 **(r,*)** Napisz klasy `kolejka_listowo` i `kolejka_tablicowo` pochodne klasy `kolejka` z zadania 2.22 zawierające odpowiednio listową i tablicową implementację kolejki. Pamiętaj o zaimplementowaniu destruktorów klas `kolejka_listowo` i `kolejka_tablicowo`.

2.24 Napisz funkcję `oproznij`, która dostaje jako argument wskaźnik do obiektu typu `kolejka` z zadania 2.22 i wypisuje w kolejnych liniach na standardowym wyjściu kolejne liczby przechowywane w kolejce aż do jej opróżnienia.

2.25 Napisz abstrakcyjną klasę `stos` definiującą interfejs stosu liczb całkowitych. Klasa `stos` powinna posiadać następujące publiczne czysto wirtualne metody:
- `z_wierzchu` zwracająca wartość elementu leżącego na wierzchu stosu,
- `usun_z_wierzchu` usuwającą element położony na wierzchu stosu,
- `dodaj_na_koniec` kładący na stosie liczbę całkowitą otrzymaną w argumencie,
- `pusty` zwracającą `true`, jeżeli stos jest pusty i `false` w przeciwnym wypadku.

Klasa powinna udostępniać także wirtualny destruktor.

2.26 **(*)** Napisz klasy `stos_listowo` i `stos_tablicowo` pochodne klasy `stos` z zadania 2.25 zawierające odpowiednio listową i tablicową implementację stosu. Pamiętaj o zaimplementowaniu destruktorów klas `stos_listowo` i `stos_tablicowo`.

2.27 **(*)** Napisz funkcję `wartosc`, która dostaje w argumencie wskaźnik do klasy `stos` z zadania 2.25 i zwraca wartość wczytanego ze standardowego wejścia wyrażenia arytmetycznego zapisanego w odwrotnej notacji polskiej. Zakładamy, że wczytywane wyrażenie składa się z oddzielonych pojedynczymi spacjami liczb całkowitych i operatorów „+", „-", „*", „/". Funkcja powinna wykorzystać w obliczeniach otrzymany w argumencie stos.

2.28 **(*)** Napisz program kalkulator. W programie wykorzystaj klasy:
- `dzialanie`, abstrakcyjną klasę posiadającą:
  - publiczne stałe pole `nazwa` przechowujące nazwę działania,
  - publiczne stałe pole `n` przechowujące arność działania,
  - czysto wirtualną publiczną metodę `wynik`, która otrzymuje jako argument wektor `n` liczb typu `double` i zwraca jako wartość wynik działania na nich.
- `kalkulator` posiadającą:
  - prywatne pole `wartosc` typu `double` zawierające obecną wartość przechowywaną w kalkulatorze,
  - wektor `dzial` referencji do obiektów typu `dzialanie`,
  - publiczną metodę `obliczenia` będącą właściwą metodą odpowiedzialną za działanie kalkulatora i komunikację z użytkownikiem. Metoda `obliczenia` w kolejnych krokach działania powinna wyświetlac bieżącą wartość pola `wartosc` oraz dawać użytkownikowi następujące możliwości do wyboru:
    - nadanie polu `wartosc` wartości wczytanej ze standardowego wejścia,
    - nadanie polu `wartosc` wartości 0,
    - wykonanie działania reprezentowanego przez jeden z obiektów, do którego referencje przechowuje wektor `dziel` i zapisanie jego wyniku do pola `wartosc`. Pierwszym argumentem działania powinna być dotychczasowa wartość pola `wartos`, pozostałe argumenty powinny zostać wczytane ze standardowego wejścia.
    - zakończenie działania metody `obliczenia,`
  - `dodaj_dzialanie`, która dostaje w argumencie referencję do obiektu klasy pochodnej klasy `dzialanie` i dodaje obiekt tej klasy do wektora `dzial`,
- `dodawanie`, `odejmowanie`, `mnozenie`, `dzielenie`, `logarytm_naturalny` klasy pochodne klasy działania.

2.29 **(*)** Napisz program do gry w kółko i krzyżyk do trzech.
- Napisz klasę `plansza`, służącą do przechowywania stanu gry. Klasa `plansza` powinna udostępniać metodę `wypisz` wypisującą na standardowym wyjściu stan planszy, metodę `stan` zwracającą stan pola

o podanych w argumentach indeksach oraz metodę `wykonaj_ruch` aktualizującą stan planszy po wykonaniu ruchu podanego jako argument.

— Napisz klasę `ruch` służącą do przechowywania pojedynczych ruchów w grze.

— Napisz klasę abstrakcyjną `gracz` posiadającą czysto wirtualną metodę `wybierz_ruch`, która dla podanych w argumentach: stałej referencji do obiektu klasy `plansza` i informacji, co ma postawić gracz (kółko czy krzyżyk), zwraca jako wartość ruch gracza.

— Napisz klasę `gracz_czlowiek` pochodną klasy `gracz`, której metoda `wybierz_ruch` dla podanej planszy zwraca ruch wczytany ze standardowego wejścia.

— Napisz klasę `gra`, która posiada prywatny obiekt typu `plansza` oraz publiczną wirtualną metodę `graj`, która dostaje jako argumenty dwa wskaźniki do obiektów `gracz`, przeprowadza grę między nimi wywołując na zmianę metodę `wybierz_ruch` obu obiektów i aktualizując stan planszy, a na koniec zwraca `1`, jeżeli wygrał `gracz` podany w pierwszym argumencie, `-1` gdy wygrał gracz podany w drugim argumencie oraz `0` w przypadku remisu. Jeżeli metoda `wybierz_ruch` któregoś z obiektów zwróci niepoprawny ruch, metoda `graj` powinna ją wywołać jeszcze raz.

2.30 **(*)** Napisz program do gry w kółko i krzyżyk pomiędzy dwojgiem ludzi z wykorzystaniem klas z zadania 2.29.

2.31 **(*)** Napisz klasę `gracz_komputer` pochodną klasy `gracz` z zadania 2.29, w której metoda `wybierz_ruch` zwraca ruch na losowo wybrane wolne pole. Napisz program do gry w kółko i krzyżyk pomiędzy człowiekiem a komputerem wykorzystując klasę `gracz_komputer` oraz klasy z zadania 2.29.

2.32 **(**)** Napisz program do gry w warcaby. Napisz klasy analogiczne do tych z zadania 2.29. Porównaj swój program z rozwiązaniem zadania 2.30.

Rozdział 3

Zaawansowane programowanie obiektowe

3.1 **(r)** Napisz klasę `stale` zawierającą publiczne stałe statyczne pola `pi` i `e`.

3.2 **(r)** Napisz klasę `liczba` zawierającą publiczne statyczne pole `licz` typu `int`. Wartość pola `licz` zainicjuj wartością 0.

3.3 Napisz klasę `tablica` zawierającą jako pole 5-elementową statyczną tablicę liczb całkowitych. Elementy tablicy zainicjuj wartościami kolejnych liczb pierwszych począwszy od 2.

3.4 Napisz klasę `stala_tablica` zawierającą jako pole 5-elementową statyczną tablicę stałych liczb całkowitych. Elementy tablicy zainicjuj wartościami kolejnych liczb pierwszych począwszy od 2.

3.5 Napisz klasę `staly_wektor` zawierającą jako pole 5-elementowy statyczny wektor stałych liczb całkowitych. Elementy wektora zainicjuj wartością 0.

Listing 3.1.

```
1 class  zespolone{
  public:
3    double re,im;
     zespolone(){}
5    zespolone(double r, double i):re(r),im(i){}
  };
```

3.6 **(r)** Napisz klasę `zesp` posiadającą publiczne statyczne metody `dodaj`, `odejmij`, `pomnoz`, `podziel`, które otrzymują jako argumenty dwa obiekty typu `zespolone` z tabelki 3.3.1 i zwracają jako wartość wynik odpowiedniego działania. Klasa `zesp` powinna posiadać także publiczną statyczną stałą `i` typu `zespolone` o wartości $\sqrt{-1}$.

3.7 **(r)** Napisz program, który wyświetla na standardowym wyjściu 100 pierwszych elementów ciągu zdefiniowanego w następujący sposób:

$$a_n = \begin{cases} \sqrt{-1} & n = 1 \\ \frac{(2 \cdot a_{n-1} + 10 \cdot \sqrt{-1})}{a_{n-1}} & n > 1 \end{cases}$$

W programie wykorzystaj klasę `zesp` z zadania 3.6.

3.8 **(r)** Stwórz przestrzeń zazw `zesp`. W tej przestrzeni nazw zdefiniuj funkcje `dodaj`, `odejmij`, `pomnoz`, `podziel`, które otrzymują jako argumenty dwa obiekty typu `zespolone` z zadania 1.21 i zwracają jako wartość wynik odpowiedniego działania. Ponadto zdefiniuj w przestrzeni nazw `zesp` stałą typu `zespolone` o wartości $\sqrt{-1}$.

3.9 **(r)** Rozwiąż zadanie 3.6 wykorzystując elementy przestrzeni nazw `zesp` z zadania 3.6.

3.10 Napisz klasę `zaokraglij`, zawierającą statyczne publiczne metody służące do zaokrąglania liczb wymiernych do liczb całkowitych. Klasa `zaokraglij` powinna udostępniać metody `najblizsza`, `podloga` i `sufit`.

3.11 Napisz program, który wczytuje ze standardowego wejścia liczby wymierne i wypisuje je po zaokrągleniu na standardowym wyjściu. W programie wykorzystaj klasę `zaokraglij` z zadania 3.10. O tym, ile liczb powinno zostać wczytanych i w jaki sposób zaokrąglonych (w górę, w dół czy do najbliższej liczby całkowitej), powinien decydować użytkownik.

3.12 Stwórz przestrzeń nazw `zaokraglij`, zawierającą funkcje służące do zaokrąglania liczb wymiernych do liczb całkowitych. Przestrzeń nazw `zaokraglij` powinna zawierać funkcje `najblizsza`, `podloga` i `sufit`.

3.13 Rozwiąż zadanie 3.11 z wykorzystaniem przestrzeni nazw `zaokraglij` z zadania 3.12.

3.14 **(r,!)** Napisz klasę `policzona` posiadającą publiczną metodę `ile` zwracającą jako wartość liczbę istniejących w danym momencie obiektów tej klasy.

3.15 Napisz klasę `unikalne`, której obiekty posiadają stałe prywatne pole `id` typu `unsigned int`. Klasę `unikalne` zaimplementuj w taki sposób, żeby każdy obiekt tego typu w programie w momencie powstawania otrzymywał inną wartość pola `id`.

3.16 **(r,!)** Klasę `zesp` z zadania 3.6 oraz klasę `zaokraglij` z zadania 3.10 napisz w taki sposób, żeby nie można było stworzyć obiektów tych klas.

3.17 **(r,!)** Napisz klasę `finalna`, po której nie można dziedziczyć.

3.18 **(r,!)** Zaimplementuj klasę `dynamiczna` przechowującą tablicę liczb całkowitych w taki sposób, żeby obiektów tej klasy nie dało się stworzyć jako zmiennych automatycznych.

3.19 **(r,!)** Zdefiniuj klasę `tab_info` służącą do przechowywania parametrów tworzonej tablicy liczb całkowitych. Klasa `tab_info` powinna przechowywać informacje o rozmiarze tworzonej tablicy, początkową wartość elementów tablicy oraz o tym, ile razy mogą być zmieniane wartości poszczególnych elementów tablicy. Obiekt klasy `tab_info` powinien także przechowywać informację, czy wartości poszczególnych parametrów były ustawiane czy nie. Klasa `tab_info` powinna udostępniać następujące publiczne metody:
   - `rozmiar`, która nadaje polu przechowującemu rozmiar tablicy wartość podaną w argumencie i zwraca jako wartość referencję do obiektu, na rzecz którego została wywołana metoda,
   - `wartosc`, która otrzymuje jako argument początkową wartość elementów tablicy i zwraca jako wartość referencję do obiektu, na rzecz którego została wywołana metoda,
   - `zmiany`, która otrzymuje jako argument liczbę dozwolonych zmian wartości poszczególnych elementów tablicy. Otrzymanie w argumen-

cie wartości -1 oznacza, że wartości elementów tablicy mogą być zmieniane dowolną liczbę razy. Metoda powinna zwrócić jako wartość referencję do obiektu, na rzecz którego została wywołana.

3.20 **(r,*)** Napisz klasę `tablica` służącą do przechowywania tablicy liczb całkowitych. Klasa ta powinna umożliwiać ograniczenie liczby zmian wartości poszczególnych elementów przechowywanej tablicy. Klasa `tablica` powinna udostępniać następujące publiczne metody:
- konstruktor, który dostaje jako argument stałą referencję do obiektu `param` typu `tab_info` z zadania 3.19, zawierającego parametry tworzonej tablicy. W przypadku niezdefiniowania któregoś z parametrów przyjmujemy jego wartość domyślną. Domyślny rozmiar tablicy, to 100, domyśla wartość początkowa elementów tablicy to 0, zaś domyślna dozwolona liczba zmian wartości każdego z elementów to 10.
- destruktor zwalniający pamięć zajmowana przez obiekt,
- `podaj_w`, która zwraca wartość elementu tablicy o indeksie podanym w parametrze,
- `nadaj_w`, która otrzymuje jako argumenty indeks `i` oraz liczbę całkowitą `w` i nadaje elementowi tablicy o indeksie `i` wartość `w`. Jeżeli element o indeksie `i` był zmieniany już maksymalną dozwoloną liczbę razy, to funkcja nie powinna nic robić.
- `licznik`, która podaje, ile jeszcze razy można zmieniać wartość elementu o indeksie `i`. Jeżeli nie ma ograniczeń na liczbę zmian wartości elementów przechowywanej tablicy, to funkcja powinna zwrócić wartość -1.
- `rozmiar` zwracającą jako wartość rozmiar tablicy.

3.21 **(r,!)** Napisz bezargumentową funkcję `alokuj`, która alokuje w pamięci obiekt typu `tablica` z zadania 3.20 przechowujący 50-elementową tablicę. Komórki alokowanej tablic powinny być wypełnione zerami, zaś wartość każdej z komórek tablicy powinna móc być zmieniona dokładnie jeden raz. Funkcja `alokuj` powinna zwrócić jako wartość wskaźnik do zaalokowanego obiektu.

3.22 Zdefiniuj klasę `punkt` przeznaczoną do przechowywania współrzędnych punktu na płaszczyźnie. Napisz klasę `opcje` przechowującą parametry wyświetlania danych punktu na standardowym wyjściu. Uwzględnij możliwość wyświetlania współrzędnych w kartezjańskim i biegunowym układzie współrzędnych oraz wyświetlania dokładnych lub zaokrąglonych wartości współrzędnych. Klasa `opcje` powinna udostępniać następujące publiczne metody:
- `kartezjanski`, która ustawia opcję wyświetlania współrzędnych w kartezjańskim układzie współrzędnych i zwraca jako wartość referencję do obiektu, na rzecz którego została wywołana,

— `biegunowy`, która ustawia opcję wyświetlania współrzędnych w biegunowym układzie współrzędnych i zwraca jako wartość referencję do obiektu na rzecz, którego została wywołana,
— `dokladnosc`, która dostaje jako argument liczbę `d` typu `double`, ustawia opcję wyświetlania współrzędnych zaokrąglonych w dół do najbliższej wielokrotności `d` i zwraca jako wartość referencję do obiektu na rzecz, którego została wywołana.

3.23 Napisz funkcję, która otrzymuje dwa argumenty obiekty: `p` typu `punkt` oraz `op` typu `opcje` i wyświetla współrzędne punktu `p` w sposób zdefiniowany w `op`. Typy `punkt` i `opcje` zostały zdefiniowane w zadaniu 3.22.

3.24 **(r,!)** Napisz klasę `napis` zawierającą publiczne pole typu `string` oraz publiczną bezargumentową metodę `stala` zwracającą wartość `true` gdy jest wywoływana na rzecz stałego obiektu i `false` w przeciwnym razie.

3.25 **(r,!)** Napisz klasę `napis2` publicznie dziedziczącą po klasie `string` i posiadającą publiczną bezargumentową metodę `stala` zwracającą wartość `true` gdy jest wywoływana na rzecz stałego obiektu i `false` w przeciwnym razie. Dla klasy `napis2` zdefiniuj następujące konstruktory:
— bezargumentowy, inicjujący obiekt przechowujący pusty napis,
— otrzymujący w argumencie zmienną typu `const string&` i inicjujący obiekt przechowujący napis otrzymany w argumencie,
— otrzymujący w argumencie zmienną typu `const char[]` i inicjujący obiekt przechowujący napis otrzymany w argumencie,

Czym w użyciu rożni się klasa `napis2` od klasy `napis` z zadania 3.24.

3.26 Napisz klasę `prywatna_liczba`, która zawiera prywatne pole `liczba` typu `int` oraz publiczne metody `wypisz` i `wczytaj`. Metodę `wypisz` zaimplementuj w taki sposób, żeby można ją było wywołać również na rzecz obiektów stałych.
Napisz funkcję `wypisz_vec`, która dostaje jako argument referencję do wektora o elementach typu `const prywatna_liczba` i wypisuje na standardowym wyjściu liczby przechowywane we wszystkich elementach wektora. Do wypisywania wartości użyj metody `wypisz`.

Listing 3.2.

```
class bazowa{
  public:
    virtual ~bazowa(){
    }
};

class pochodna: public bazowa{
};
```

3.27 **(r)** Napisz funkcję `porownaj`, która otrzymuje jako argumenty dwa wskaźniki typu `const bazowa *`, gdzie typ `bazowa` został zdefiniowany w listingu 3.2. Funkcja powinna zwrócić jako wartość `true`, jeżeli wskazywane przez argumenty obiekty są tego samego typu oraz `false` w przeciwnym wypadku.

3.28 **(r)** Napisz funkcję, która otrzymuje jako argument wskaźnik typu `bazowa *` i zwraca wartość typu `pochodna *`, gdzie typy `bazowa` i `pochodna` zostały zdefiniowane w listingu 3.2. Jeżeli argument funkcji wskazuje na obiekt typu `pochodna`, to funkcja powinna zrzutować go na typ `pochodna *` i zwrócić jako wartość. W przeciwnym wypadku funkcja powinna zwrócić wartość `NULL`.

Listing 3.3.

```
class liczba{
  public:
    virtual void wypisz()=0;
};

class rzeczywista: public liczba{
  public:
    double wartosc;
    void wypisz(){
        cout<<wartosc<<endl;
    }
};

class zespolona: public liczba{
  public:
    double wartR, wartU;
    void wypisz(){
        cout<<wartR<<" "<<wartU<<endl;
    }
};
```

3.29 Napisz funkcję `wypisz`, która dostaje jako argumenty tablicę o elementach typu `liczba*` i jej rozmiar, która wywołuje metodę `wypisz` na rzecz obiektów typu `rzeczywista` wskazywanych przez elementy otrzymanej w argumencie tablicy (elementy tablicy wskazujące na obiekty innych typów powinny zostać zignorowane). Typy `liczba` i `rzeczywista` zostały zdefiniowane w listingu 3.3

Listing 3.4.

```
class liczba2{
  public:
    virtual ~liczba2(){}
};
```

```
 6 class rzeczywista2: public liczba2{
     public:
 8     double wartosc;

10 };

12 class zespolona2: public liczba2{
     public:
14     double wartR, wartU;
   };
```

3.30 **(r)** Napisz funkcję `wypisz`, która dostaje jako argumenty tablicę o elementach typu `liczba2*` i jej rozmiar, która wypisuje na standardowe wyjście wartości przechowywane w obiektach typu `rzeczywista2` wskazywanych przez elementy otrzymanej w argumencie tablicy (elementy tablicy wskazujące na obiekty innych typów powinny zostać zignorowane). Typy `liczba2` i `rzeczywista2` zostały zdefiniowane w listingu 3.4

Rozdział 4

Przeciążanie operatorów

4.1 **(r)** Napisz klasę `zespolone` służącą do przechowywania liczb zespolonych. Przeciąż dla niej operatory odpowiadające działaniom arytmetycznym oraz operatory `<<` i `>>` tak, żeby wartości typu `zespolone` można było wczytywać i wypisywać przy pomocy standardowych strumieni.

4.2 Napisz klasę `n_int` posiadającą dwa prywatne pola: `liczba` typu `int` oraz `okr` typu `bool`. Wartość pola `okr` powinna być `true` gdy wartość pola `liczba` jest określona. Klasa `n_int` powinna udostępniać:
- bezargumentowy konstruktor nadający polu `okt` wartość `false`,
- konstruktor nadający polu `liczba` wartość otrzymaną w argumencie, zaś polu `okr` wartość `true`,
- przeciążone operatory arytmetyczne. W przypadku gdy wartość któregoś z argumentów operatora nie jest określona (jego pole `okr` ma wartość `false`), wartość wyniku również powinna być nieokreślona.
- operator `>>` przeciążony w taki sposób, żeby umożliwiał wczytywanie ze standardowego wejścia wartości pola `liczba` (po pomyślnym wczytaniu wartości pola `liczba` pole `okr` powinno otrzymać wartość `true`),
- operator `<<` przeciążony w taki sposób, żeby umożliwiał wypisywanie na standardowym wyjściu wartości pola `liczba`. W przypadku, gdy pole `okr` ma wartość `false`, nic nie powinno zostać wypisane.

4.3 **(*)** Napisz klasę `d_int` służącą do przechowywania dużych liczb całkowitych (niemożliwych do przechowania w typach standardowych). Obiekty klasy `d_int` powinny przechowywać liczby jako tablice zmiennych typu `unsigned char`. Klasa `d_int` powinna posiadać:
- konstruktor bezargumentowy inicjujący obiekt przechowujący wartość 0,
- konstruktor inicjujący obiekt przechowujący liczbę podaną w argumencie,
- konstruktor kopiujący,
- destruktor
- przeciążone operatory porównania (`==`, `<`),
- przeciążone operatory arytmetyczne,
- przeciążony operator przypisania,
- przeciążone operatory `<<` i `>>` tak, żeby umożliwić proste wczytywanie i wypisywanie na standardowym wyjściu zmiennych typu `d_int`.

4.4 **(r,*)** Napisz klasę `tablica` służącą do przechowywania dynamicznych tablic liczb całkowitych. Klasa `tablica` powinna posiadać prywatne pola `tab` typu `int *` oraz `rozmiar` typu `unsigned int` i udostępniać:
- konstruktor bezargumentowy inicjujący obiekt nie przechowujący

tablicy (inicjujący pole `tab` wartością `NULL`, a pole `rozmiar` wartością 0),
- konstruktor tworzący tablicę o liczbie elementów podanej w argumencie,
- konstruktor kopiujący,
- destruktor,
- metodę `resize` zmieniającą rozmiar przechowywanej tablicy do rozmiaru podanego w argumencie. Zawartość starej tablicy powinna zostać przepisana na początek nowej tablicy. W przypadku zmniejszenia rozmiaru tablicy, powinno być przepisane tyle elementów starej tablicy, ile się zmieści w nowej.
- operator przypisania,
- operator `[]` zwracający referencję do poszczególnych elementów tablicy (powinien to być jedyny sposób dostępu z zewnątrz klasy do elementów tablicy). Przeciąż ten operator zarówno w wersji dla zmiennych jak i stałych obiektów.

4.5 **(*)** Napisz klasę `wektor` służącą do przechowywania wektorów. Klasa `wektor` powinna zawierać prywatne pola `tab` typu `double` oraz `rozmiar` typu `unsigned int`. Ponadto klasa `wektor` powinna posiadać:
- konstruktor `wektor(int n)` tworzący n-wymiarowy wektor,
- konstruktor kopiujący,
- przeciążone operatory dodawania wektorów i mnożenia wektorów przez skalar,
- przeciążony operator przypisania,
- przeciążone operatory porównania `==` oraz `<=` (jeden wektor jest mniejszy lub równy od drugiego jeżeli jest mniejszy lub równy na wszystkich współrzędnych).
- przeciążony operator `[]` zwracający referencję do współrzędnej wektora o podanym w argumencie indeksie (przeciąż ten operator w wersji dla obiektów zmiennych i stałych).

4.6 **(r)** Napisz klasę `napis` służącą do przechowywania napisów. Klasa powinna zawierać:
- bezargumentowy konstruktor tworzący pusty napis,
- konstruktor inicjujący obiekt wartością tablicy znaków (`char *`) otrzymanej w argumencie. Zakładamy, że otrzymany w argumencie napis zakończony jest znakiem o numerze 0 (tak jak to jest w napisach w języku C),
- konstruktor kopiujący,
- destruktor,
- przeciążony operator przypisania,
- przeciążony operator `==`,

— przeciążony operator `[]`, zwracający wartości poszczególnych znaków napisu,
— przeciążony operator `+`, który ma działać jak operator katenacji (ma zwracać jako wynik sklejony napis). Zaimplementuj ten operator tak aby dało się go składać z samym sobą,
— metodę `rozmiar` zwracającą liczbę znaków zawartych w przechowywanym napisie,
— metodę `wypisz` wypisującą przechowywany napis na standardowym wyjściu.

4.7 **(!)** Napisz funkcję `niekopiowalna` w taki sposób, żeby obiektów tej klasy nie dało się kopiować ani za pomocą konstruktora kopiującego, ani za pomocą operatora przypisania.

4.8 **(r,!,*)** Napisz abstrakcyjną klasę `komperator` posiadającą jedynie czysto wirtualny operator `()` o dwóch argumentach typu `const napis&` z zadania 4.6. Napisz klasę `alfabetyczna` pochodną klasy `kompertor` służącą do porównywania obiektów klasy `napis` z zadania 4.6. Przeciąż w klasie `alfabetyczna` operator `()` w taki sposób, żeby zwracał on wartość `true` jeżeli pierwszy argument jest referencją do wcześniejszego w kolejności alfabetycznej napisu i `false` w przeciwnym wypadku.

4.9 **(r,!,*)** Napisz funkcję `sortuj`, która otrzymuje jako argument tablicę elementów typu `napis` z zadania 4.6, rozmiar tablicy, oraz referencję do obiektu klasy `komperator` 4.8 i sortuje rosnąco podaną w argumencie tablicę używając do porównywania elementów tablicy otrzymanego w drugim argumencie obiektu klasy `komperator`.

4.10 Napisz program, który tworzy tablicę o elementach typu `napis` z zadania 4.6, nadaje jej elementom wartości wczytane ze standardowego wejścia, sortuje elementy tablicy przy pomocy funkcji `sortuj` z zadania 4.9 z argumentem typu `alfabetyczna` zdefiniowanego w zadaniu 4.8 i wypisuje posortowane elementy tablicy na standardowym wyjściu.

4.11 **(!,*)** Napisz klasę służącą do generowania liczb pseudolosowych. Klasa ta powinna przeciążony operator `()`, który powinien zwracać dwa rodzaje wartości pseudolosowych:
— liczbę całkowitą z zakresu `[0,n-1]`, gdzie `n` jest argumentem podanym przez użytkownika,
— liczbę z zakresu `(0,1)`, jeżeli użytkownik nie poda żadnego argumentu.

Klasa powinna posiadać konstruktor, którego argument powinien inicjować generator. Do generowania kolejnych wartości z zakresu `(0,1)` użyj funkcji `f(x)=1-x*x`.

4.12 **(r,!)** Napisz klasę `macierz` służącą do przechowywania tablic dwuwymiarowych o elementach typu `double`. Klasa `macierz` powinna udostępniać:

— konstruktor inicjujący macierz o podanych w argumentach wymiarach,
— przeciążony operator `()`, który otrzymuje jako argumenty dwie nieujemne liczby całkowite i zwraca jako wartość referencję do elementu przechowywanej w obiekcie tablicy o indeksach podanych w argumentach.

Dlaczego w zadaniu każemy przeciążyć operator `()`, a nie `[]`?

4.13 Stwórz typ służący do przechowywania wartości logicznych logiki trójwartościowej (prawda, fałsz, wartość nieznana). Dla zdefiniowanego typu przeciąż operatory logiczne tak, aby działały zgodnie z następującymi tabelkami:

a && b	prawda	wart. nieznana	fałsz
prawda	prawda	wart. nieznana	fałsz
wart. nieznana	wart. nieznana	wart. nieznana	fałsz
fałsz	fałsz	fałsz	fałsz

a || b	prawda	wart. nieznana	fałsz
prawda	prawda	prawda	prawda
wart. nieznana	prawda	wart. nieznana	wart. nieznana
fałsz	prawda	wart. nieznana	fałsz

a	$!a$
prawda	fałsz
wart. nieznana	wart. nieznana
fałsz	prawda

4.14 **(r)** Napisz struktury `punkt2D` i `punkt3D` służące do przechowywania współrzędnych punktów w przestrzeniach odpowiednio dwu- i trójwymiarowych. Przeciąż operator rzutowania w taki sposób, żeby można było zrzutować obiekty typu `punkt3D` na typ `punkt2D`. Rzutowanie powinno „obcinać" trzecią współrzędną punktu.

4.15 Zaimplementuj dla klasy `zespolone` z zadania 4.1 operator rzutowania na typ `double`. Operator ten powinien zwracać część rzeczywistą rzutowanej liczby.

4.16 Zaimplementuj dla klasy `d_int` z zadania 4.3 operator rzutowania na typ `int`. W przypadku gdy wartość typu `d_int` jest zbyt duża lub zbyt mała by ją zapisać w zmiennej typu `int` operator powinien zwrócić wartość `INT_MAX` lub `INT_MIN` w zależności czy rzutowana wartość jest dodatnia czy ujemna.

4.17 Dla typu zdefiniowanego w zadaniu 4.13 przeciąż operator rzutowania do typu `bool`. Operator powinien rzutować wartość „prawda" na wartość `true`, zaś wartość „nieokreśloną" i wartość „fałsz" na `false`.

4.18 **(r,!,*)** Dla struktury

```
1 struct elisty{
  struct element{
3   element * nastepny, poprzedni;
    int i;
5 };

7 element * wsk, *pierwszy, *ostatni;

9 };
```

przeciąż operatory:

— `++` przesuwający wskaźnik `wsk` na następny element listy (nadający mu wartość pola `nastepny` wskazywanej struktury). W przypadku gdy `wsk` wskazuje na ostatni element listy operator nie powinien zmieniać jego wartości.

— `--` przesuwający wskaźnik `wsk` na wcześniejszy element listy (nadający mu wartość pola `poprzedni` wskazywanej struktury). W przypadku gdy `wsk` wskazuje na pierwszy element listy operator nie powinien zmieniać jego wartości.

Przeciąż prefiksową i postfiksową wersję powyższych operatorów. Wersje prefiksowe przeciążanych operatorów powinny zwracać jako wartość referencję do otrzymanego w argumencie obiektu, a wersja postfiksowe kopię pierwotnej wartości obiektu.

4.19 Utwórz typ `iterator` umożliwiający dostęp do poszczególnych komórek tablicy o elementach typu `int`. Typ `iterator` powinien zawierać prywatne pole `wskaznik` typu `int *` oraz posiadać:

— konstruktor, który ustawia wartość pola `wskaźnik` na wartość podaną w argumencie,

— przeciążony unarny operator `*` zwracający jako wartość referencję do liczby całkowitej wskazywanej przez pole `wskaznik`,

— przeciążone operatory inkrementacji `++` w taki sposób, żeby inkrementowały wartość pola `wskaznik`,

— przeciążone operatory dekrementacji `--` w taki sposób, żeby odejmowały jedne od wartości pola `wskaznik`,

— przeciążone operatory porównania `==` oraz `<` w taki sposób, żeby porównywały wartość pól `wskaznik` porównywanych obiektów.

4.20 **(!)** Do klasy `tablica` z zadania 4.4 dopisz metody publiczne:

— `poczatek` zwracającą obiekt typu `iterator` z zadania 4.19, w którym pole `wskaznik` wskazuje na pierwszą komórkę tablicy wskazywanej przez pole `tab` obiektu, na rzecz którego została wywołana metoda.

— `koniec` zwracającą obiekt typu `iterator` z zadania 4.19, w któ-

rym pole `wskaznik` wskazuje na komórkę pamięci znajdującą się tuż za ostatnią komórką tablicy wskazywanej przez pole `tab` obiektu, na rzecz którego została wywołana metoda.

4.21 **(r, !,*)** Napisz klasę `n_int` zawierającą publiczne pole typu `int`. Klasa `n_int` powinna zawierać statyczną metodę `wypisz` wypisującą liczbę dynamicznie zaalokowanych pojedynczych obiektów typu `n_int` (nie licząc tych wchodzących w skład dynamicznych tablic) oraz liczbę dynamicznie zaalokowanych tablic o elementach typu `n_int`. Aby to umożliwić, przeciąż w odpowiedni sposób operatory `new` i `delete`.

4.22 **(!,*)** Napisz klasę `n_char` zawierającą publiczne pole typu `char`. Przeciąż operatory `new` i `delete` dla tego typu w taki sposób, żeby wszystkie dynamicznie alokowane obiekty typu `n_char` znajdowały się w jednym wcześniej zaalokowanym obszarze pamięci. Zakładamy, że liczba znajdujących się jednocześnie w pamięci obiektów typu `n_char` nie przekracza 1000. Uniemożliw dynamiczne alokowanie tablic obiektów typu `n_char`.

Rozdział 5

Szablony

W języku polskim często zamiennie używa się pojęć argument i parametr na określenie argumentów funkcji. Aby uniknąć nieporozumień w tym rozdziale i w całym skrypcie, konsekwentnie używamy pojęć argumentów funkcji i parametrów szablonu.

5.1 **(r)** Napisz szablon funkcji, która otrzymuje jako argumenty zmienne a,b,c o typie będącym parametrem szablonu i zwraca wartość a-b+c.

5.2 Napisz szablon funkcji, która wczytuje ze standardowego wejścia wartość zmiennej o typie będącym parametrem funkcji i zwraca ją jako wartość funkcji.

5.3 Napisz szablon funkcji, która otrzymuje jako argumenty dwie wartości o typie podanym jako parametr szablonu i zwraca wartość mniejszego z otrzymanych argumentów.

5.4 **(r)** Napisz szablon funkcji, która otrzymuje jako argumenty tablicę o elementach typu podanego jako parametr szablonu oraz rozmiar tablicy i zwracającą jako wartość element tablicy o najmniejszej wartości.

5.5 **(r)** Napisz szablon funkcji, która otrzymuje jako argumenty dwie referencje do zmiennych typu będącego parametrem szablonu i zamienia wartościami zmienne, do których referencje otrzymała w argumentach.

5.6 **(r)** Napisz szablon funkcji `wypisz`, która dostaje jako argument `wsk` wskaźnik wskazujący na wartość o typie będącym parametrem szablonu i wypisującą na standardowym wyjściu wartość wskazywaną przez argument.

5.7 **(r)** Napisz szablon funkcji `wypisz_zakres`, która otrzymuje dwie zmienne `pocz` i `kon`, których typ jest parametrem szablonu i wywołuje funkcję `wypisz` z zadania 5.6 dla każdego elementu z zakresu od `pocz` do `kon-1`. Zakładamy, że kolejne elementy zakresu można otrzymać poprzez inkrementację zmiennej `pocz`.

5.8 **(r,!)** Napisz szablon funkcji, która otrzymuje jako argumenty tablicę o elementach typu będącego parametrem szablonu oraz rozmiar tablicy i wypisuje wszystkie elementy otrzymanej w argumencie tablicy przy pomocy funkcji `wypisz_zakres` z zadania 5.7.

5.9 **(r)** Rozwiąż zadania od 5.6 do 5.8 w taki sposób, żeby parametrami były typy argumentów (a nie typy wskazywane przez argumenty).
Napisz szablon funkcji `wypisz_vec`, która otrzymuje jako argumenty stałą referencję do wektora (typ `vector`) o elementach typu będącego parametrem szablonu i wypisuje wszystkie elementy otrzymanego w argumencie wektora przy pomocy funkcji `wypisz_zakres` z zadania 5.7.

5.10 **(r,!)** Napisz szablon operatora `>`, który działa dla wszystkich klas, dla których zdefiniowany jest operator `<`. Operator `>` powinie zwracać `true` wtedy i tylko wtedy gdy operator `<` zwraca `false`.

5.11 **(r)** Napisz szablon dwuargumentowej funkcji, której argumenty są typów `T1` i `T2` będących parametrami szablonu, która zwraca `true` wtedy i tylko wtedy, gdy oba argumenty są tego samego typu. Zakładamy, że `T1` i `T2` są klasami zawierającymi metody wirtualne .

5.12 **(r)** Napisz szablon funkcji `minimum` o dwóch parametrach `T1` i `T2`. Zakładamy, że klasa `T2` ma przeciążony operator `()`, który dla dwóch argumentów typu `T1` zwraca wartość `true` wtedy i tylko wtedy, gdy pierwszy z argumentów jest mniejszy. Argumentami funkcji `minimum` powinny być tablica o elementach typu `T1`, rozmiar tablicy oraz obiekt typu `T2`. Funkcja powinna zwrócić wartość najmniejszego elementu otrzymanej w pierwszym argumencie tablicy względem porządku wyznaczonego przez ostatni argument.

5.13 **(r,!)** Napisz funkcję, która dostaje jako argumenty tablicę liczb całkowitych oraz jej rozmiar i wypisuje na standardowym wyjściu najmniejszy i największy element otrzymanej w argumencie tablicy. Do znalezienia największego i najmniejszego elementu użyj funkcji `minimum` z zadania 5.12.

5.14 Napisz szablon funkcji `dla_kazdego` o dwóch parametrach `T1` i `T2`. Zakładamy, że klasa `T2` ma przeciążony operator `()` tak, że jego argumentem może być zmienna typu `T1`. Argumentami funkcji `dla_kazdego` powinny być tablica `t` o elementach typu `T1`, rozmiar tablicy oraz stała referencja `f` do obiektu typu `T2`. Funkcja powinna wywołać operator `()` na rzecz obiektu `f` dla wszystkich elementów tablicy `t`.

5.15 Napisz program, który wczytuje ze standardowego wejścia tablicę napisów, zamienia wszystkie małe litery w napisach na duże i wypisuje na standardowym wyjściu efekty swojej pracy. Wykorzystaj do przetwarzania tablicy napisów szablon funkcji `dla_kazdego` z zadania 5.14.

5.16 **(r)** Napisz szablon funkcji `przepisz` o jednym parametrze `n` typu `unsigned int`, która dostaje jako argumenty dwie tablice dwuwymiarowe `n` na `n` typu `int[n][n]` i przepisuje zawartość tablicy otrzymanej w pierwszym argumencie do tablicy otrzymanej w drugim argumencie.

5.17 **(r)** Napisz klasę `tablica` służącą do przechowywania tablicy liczb typu `int`. Klasa `tablica` powinna posiadać publiczne pola `tab` typu `int *` oraz `rozmiar_tab` typu `unsigned int` i udostępniać:

- konstruktor, który otrzymuje jako argument dodatnią liczbę całkowitą `n`, tworzy `n`-elementową tablicę o elementach typu `int`, przypisuje do pola `tab` wskaźnik do nowo utworzonej tablicy i przypisuje polu `rozmiar` wartość `n`,
- destruktor,
- szablon metody `sortuj`, której parametrem jest typ `T` o operatorze `()` przeciążonym w taki sposób, że dla dwóch liczb podanych w argumencie zwraca `true` wtedy i tylko wtedy, gdy pierwsza liczba

jest mniejsza według pewnego ustalonego porządku. Metoda `sortuj` powinna otrzymywać w argumencie obiekt typu `T` i sortować elementy przechowywanej tablicy zgodnie z porządkiem wyznaczonym przez operator `()` otrzymanego w argumencie obiektu.

5.18 Napisz klasę `obudowany_vector`, która posiada prywatne pole `wektor` typu `vector<double>` oraz publiczny szablon metody `kopiuj`, który dostaje jako argument stałą referencję do obiektu typu `vector<T>` i przekopiowuje zawartość otrzymanego w argumencie wektora do pola `wektor`. Zakładamy, że `T` może być dowolnym standardowym typem liczbowym.

5.19 Napisz klasę `statystyka`, która posiada:
- prywatne pola `maksimum`, `minimum` i `srednia` typu `double`,
- publiczne metody `ZwrocMaks`, `ZwrocMin`, `ZwrocSred` zwracające wartość pól odpowiednio `maksimum`, `minimum` i `srednia`,
- szablon metody `wczytaj` otrzymujący jako argument zmienną `wek` typu `const vector<T>&`, gdzie `T` jest parametrem szablonu i nadający polom `maksimum`, `minimum` i `srednia` odpowiednio minimalną, maksymalną i średnią wartość elementów wektora `wek`.

5.20 W klasie `statystyka` z zadania 19 przeciąż szablon metody `wczytaj` tak, żeby działał także dla podanych w argumencie dwuwymiarowych kwadratowych tablic automatycznych o elementach standardowych typów liczbowych i różnych rozmiarach.

5.21 Napisz klasę `plik` służącą do zapisywania danych do plików tekstowych. Klasa `plik` powinna posiadać:
- konstruktor otwierający do pisania plik, do którego ścieżkę dostępu otrzymał w argumencie,
- prywatne pole przechowujące strumień skojarzony z otworzonym plikiem,
- szablon metody `zapisz` zapisujący do pliku zawartość dwuwymiarowej automatycznej tablicy kwadratowej o typie elementów oraz wymiarach będących parametrami szablonu. Zakładamy, że dla typu elementów tablicy przeciążony jest operator `<<` umożliwiający pisanie do strumienia. Wszystkie elementy jednego wiersza tablicy powinny zostać zapisane w jednym wierszu pliku oddzielone pojedynczymi odstępami. Każdy wiersz tablicy powinien być zapisany w oddzielnym wierszu pliku.

5.22 **(r)** Napisz szablon klasy `liczba`, posiadającego prywatne pole `wart` typu będącego parametrem szablonu. Klasa `liczba` powinna posiadać następujące publiczne metody:
- `wczytaj` wczytującą ze standardowego wejścia wartość pola `wart`,
- `wartosc` zwracającą wartość pola `wart`.

5.23 Napisz szablon klasy `tablica` o jednym parametrze `n` typu `unsigned int`.

Klasa `tablica` powinna posiadać publiczne pole będące `n`-elementową automatyczną tablicę o elementach typu `int`.

5.24 Napisz szablon funkcji `przepisz` o jednym parametrze `n` typu `unsigned int`, która dostaje jako argumenty zmienną `a` typu `tablica<n>&` i `b` typu `const tablica<n>&`, i przepisuje zawartość obiektu, do którego referencję przechowuje `b` do zmiennej, do której referencję zawiera `a`. Szablon `tablica` został zdefiniowanych w zadaniu 5.33.

5.25 **(r)** Napisz szablon klasy `para` o dwóch parametrach `T1` i `T2`, który posiada dwa pola publiczne `pierwsze` typu `T1` i `drugie` typu `T2`. Zdefiniuj dla klasy `para`:
- dwuargumentowy konstruktor nadający polom tej klasy wartości podane w argumentach,
- operator `<` w taki sposób, że element `a` jest mniejszy od elementu `b` wtedy i tylko wtedy, gdy `a.pierwsze<b.pierwsze` lub gdy `a.pierwsze=b.pierwsze` i `a.drugie<b.drugie`.

5.26 Napisz szablon `tablica` posiadającej publiczne pola `tab` typu `T*`, gdzie typ `T` jest parametrem szablonu oraz stałe pole `rozmiar` typu `unsigned int`. Klasa `tablica` powinna posiadać konstruktor, który otrzymuje w argumencie zmienną `n` typu `unsigned int`, przypisuje wartość `n` stałej `rozmiar`, tworzy dynamicznie `n`-elementową tablicę o elementach typu `T` i przypisuje wskaźnik do niej do pola `tab`. Zdefiniuj dla typu `tablica` konstruktor kopiujący, destruktor oraz operator przypisania.

5.27 **(r,!)** Napisz szablon klasy `tablica2` posiadający dwa parametry: typ `T` i wartość `n` typu `unsigned int`. Klasa tablica powinna posiadać jako publiczne pole `n`-elementową automatyczną tablicę `tab` o elementach typu `T` i statyczną stałą `rozmiar` typu `int`, która powinna zostać zainicjowana wartością `n`. Czy powinniśmy zdefiniować dla typu `tablica2` konstruktor kopiujący, destruktor albo operator przypisania?

5.28 **(r,!)** Napisz szablon klasy `wektor` posiadający jako parametr `n` typu `unsigned int`. Klasa `wektor` powinna posiadać jako publiczne pole `n`-elementową automatyczną tablicę o elementach typu `double` przechowująca współrzędne wektora. Dla klasy wektor przeciąż operatory dodawania i odejmowania wektorów oraz mnożenia wektora przez skalar (wartość typu `double`).

5.29 **(r,!)** Napisz szablon klasy `punkt` posiadający jeden parametr `n` typu `unsigned int`, służący do przechowywania współrzędnych punktów w `n`-wymiarowym kartezjańskim układzie współrzędnych. Zakładamy, że współrzędne są przechowywane w publicznej `n`-wymiarowej automatycznej tablicy o elementach typu `double`.

5.30 **(r,!)** Napisz szablon funkcji `zrzutuj` o jednym parametrze `n`, typu `unsigned int`, która otrzymuje jako argument stałą referencję do obiek-

tu typu `punkt<n>` z zadania 5.29 i zwraca jako wartość obiekt typu `punkt<n-1>` powstały z argumentu funkcji poprzez pominięcie ostatniej współrzędnej.

5.31 **(r,!)** Do szablonu klasy `punkt` z zadania 5.29 dopisz konstruktor, który dostaje jako argument stałą referencję `p` do obiektu klasy `punkt<n+1>` i przepisuje do tworzonego punktu wszystkie współrzędne `p` poza ostatnią. Czym różnią się w użyciu: zdefiniowany tym w tym zadaniu konstruktor i zdefiniowana w poprzednim zadaniu funkcja `zrzutuj`.

5.32 Do szablonu klasy `punkt` z zadania 29 dopisz szablon konstruktora o parametrze `m`, który dostaje jako argument stałą referencję `p` do obiektu klasy `punkt<m>` i:
- jeżeli `m` jest większe lub równe `n`, to przepisuje do tworzonego punktu pierwsze `n` współrzędnych punktu `p`,
- jeżeli `m` jest mniejsze od `n`, to pierwszym `m` współrzędnym nowo tworzonego punktu przypisuje wartości współrzędnych z punktu `p`, natomiast pozostałym współrzędnym nadaje wartość 0.

5.33 Napisz szablon klasy `macierz`, która posiada dwa parametry `m` i `n` typu `unsigned int`. Klasa `macierz` powinna przechowywać publiczną dwuwymiarową automatyczną tablicę o wymiarach `m` na `n` i elementach typu `int`.

5.34 Napisz szablon funkcji `kopiuj` o dwóch parametrach `n` i `m` typu `unsigned int`, która otrzymuje jako argumenty zmienną `a` typu `macierz<n,m>&` i `b` typu `const macierz<n,m>&`, i przepisuje zawartość obiektu do którego referencję przechowuje `b` do zmiennej, do której referencję zawiera `a`. Szablon `macierz` został zdefiniowanych w zadaniu 5.33.

5.35 Przeciąż operator `*` dla szablonu klasy `macierz` z zadania 5.33. Operator `*` powinien zwracać jako wartość iloczyn macierzy otrzymanych w argumentach.

5.36 Zdefiniuj szablon klasy `porownywacz` o jednym parametrze `T` i przeciążonym operatorze `()`. Klasę `porownywacz` zdefiniuj w taki sposób, żeby mogła służyć jako drugi parametr (a obiekt tej klasy jako trzeci argument) w szablonie funkcji `minimum` z zadania 5.12. Operator `()` przeciąż w taki sposób, żeby funkcja `minimum` zwracała jako wartość największy, zgodnie z porządkiem na wartościach `T` określonych przez operator `<`, element otrzymanej w argumencie tablicy .

5.37 Zdefiniuj szablon klasy `prownywacz2` o jednym parametrze `T` i przeciążonym operatorze `()`. Klasę `porownywacz2` zdefiniuj w taki sposób, żeby mogła służyć jako drugi parametr (a obiekt tej klasy jako trzeci argument) w szablonie funkcji `minimum` z zadania 5.12. Klasa `porownywacz2` powinna posiadać konstruktor o argumencie typu `bool`, którego wartość powinna decydować o sposobie działania operatora `()`. W obiektach

utworzonych z podaniem jako argumentu konstruktora wartości `true` operator `()` powinien działać jak operator `<`. Podanie do konstruktora wartości `false` powinno sprawić, że operator `()` działa jak operator `>`.

5.38 **(r,*)** Napisz szablon abstrakcyjnej klasy `lista`, którego parametrem jest typ `T` elementów przechowywanych w liście. Szablon klasy `lista` powinien udostępniać następujące czysto wirtualne publiczne metody (opisy dotyczą zachowania poszczególnych metod w klasach pochodnych klasy `lista`):
- `wstaw_z_przodu` wstawiającą na początek listy element podany w argumencie metody,
- `pierwszy` zwracającą wartość pierwszego elementu listy,
- `usun_pierwszy` usuwającą pierwszy element listy,
- `wstaw_z_tylu` wstawiającą na koniec listy element podany w argumencie metody,
- `ostatni` zwracającą wartość ostatniego elementu listy,
- `usun_ostatni` usuwającą ostatni element listy,
- `pusta` zwracającą `true` gdy lista jest pusta i `false` w przeciwnym wypadku.

Pamiętaj o zdefiniowaniu w klasie `lista` wirtualnego destruktora.

5.39 **(r,*)** Napisz szablon klasy `lista_wskaz` pochodnej klasy `lista` z zadania 5.38. Klasa `lista_wskaz` powinna przechowywać elementy listy w liście wskaźnikowej. Klasa `lista_wskaz` powinna posiadać bezargumentowy konstruktor oraz destruktor.

5.40 Napisz szablon klasy `lista_tab` pochodnej klasy `lista` z zadania 5.38. Klasa `lista_tab` powinna przechowywać elementy listy w powiększanej w miarę potrzeby tablicy. Klasa `lista_tab` powinna posiadać bezargumentowy konstruktor oraz destruktor.

5.41 Napisz szablon klasy `stos` będący implementacją stosu elementów o typie podanym jako parametr szablonu. Klasa `stos` powinna wykorzystywać szablon klasy `lista_wskaz` z zadania 5.38 do przechowywania stosu. Klasa `stos` powinna udostępniać następujące metody publiczne:
- `z_wierzchu` zwracającą jako swoją wartość element znajdujący się na wierzchu stosu (pierwszy na liście),
- `usun_z_wierzchu` usuwający element znajdujący się na wierzchu stosu,
- `poloz_na_stos` kładący na stosie (wstawiający na początek listy) element o wartości podanej w argumencie,
- `pusty` zwracającą `true` jeżeli stos jest pusty i `false` w przeciwnym wypadku.

5.42 **(r)** Napisz szablon klasy `kolejka` o jednym parametrze `T`. Szablon `kolejka` powinien być implementacją kolejki o elementach typu `T` prze-

chowywanych w obiekcie klasy pochodnej klasy `lista` z zadania 5.38. Klasa `kolejka` powinna udostępniać następujące metody publiczne:
- konstruktor, który w argumencie otrzymuje wskaźnik do obiektu klasy pochodnej klasy `lista` z zadania 5.38, w którym mają być przechowywane elementy kolejki,
- bezargumentowy konstruktor inicjujący kolejkę przechowującą swoje elementy w obiekcie typu `lista_wskaz` z zadnia 5.39,
- `pierwszy` zwracająca wartość pierwszego elementu kolejki,
- `usun_pierwszy` usuwającą pierwszy element kolejki,
- `dodaj_na_koniec`, dodającą na koniec kolejki wartość otrzymaną w argumencie
- `pusta` zwracającą `true` jeżeli kolejka jest pusta i `false` w przeciwnym wypadku.

5.43 **(r)** Napisz szablon klasy `tablica` o jednym parametrze typie `T`. Klasa `tablica` powinna posiadać prywatne pola `tab` typu `T *` i `roz` typu `unsigned int` oraz udostępniać:
- bezargumentowy konstruktor przypisujący polu `roz` wartość 0.
- konstruktor, który dostaje w argumencie wartość `n`, tworzy `n`-elementową tablicę o elementach typu `T`, przypisuje do pola `tab` wartość wskaźnika do nowo utworzonej tablicy i nadaje polu `roz` wartość `n`.
- konstruktor kopiujący,
- destruktor,
- bezargumentową metodę `rozmiar` zwracającą wartość pola `roz`,
- przeciążony operator przypisania.
- przeciążony operator `[]`, który dla podanego indeksu `i` zwraca referencję do komórki tablicy `tab` o indeksie `i`,
- operator `[]` przeciążony jako stała metoda i zwracający stałą referencję do elementu tablicy `tab` o podanym w argumencie indeksie.

5.44 **(r)** Stwórz klasę `napis` będącą konkretyzacją szablonu `tablica` z zadania 5.43 dla typu `char`. Dla klasy `napis` przeciąż operator `+` tak, żeby działał jak operator katenacji.

5.45 **(*)** Dla klasy `napis` z zadania 5.44 przeciąż operatory `<<` i `>>` w taki sposób, żeby umożliwić wczytywanie ze standardowego wejścia i wypisywanie na standardowym wyjściu wartości obiektów klasy `napis` przy pomocy strumieni `cin` i `cout`.

5.46 **(r,!,*)** Dla szablonu klasy `tablica` z zadania 5.43 przeciąż operator porównania `==` w taki sposób, żeby zwracał `true` wtedy i tylko wtedy, gdy dwie tablice mają ten sam rozmiar i taką samą zawartość. Operator porównania przeciąż jako metodę szablonu klasy `tablica`. Przeciąż go tak, by można nim było porównywać obiekty typów będących konkretyzacją szablonu `tablica` różnymi porównywalnymi typami (czyli

przechowujących tablice o elementach różnych typów, których wartości można porównywać np. tablice o elementach typu `int` i `double`).

5.47 **(*)** Napisz szablon klasy `macierz` o dwóch parametrach `n` i `m` typu `int` wyznaczających wymiary macierzy. Szablon klasy `macierz` powinien posiadać publiczne pole `M` będące automatyczną tablicą o elementach typu `double` i wymiarach `n` na `m`. Napisz szablon metody `pomnoz` otrzymującej jako argument stałą referencję do obiektu typu `macierz` i zwracającą jako wartość wynik mnożenie obiektu, na rzecz którego została wywołana metoda i macierzy otrzymanej w argumencie metody.

5.48 **(r,!)** Napisz program wczytujący ze standardowego wejścia pewną podaną przez użytkownika liczbę wartości typu `int` i wypisującą ją w kolejności od najmniejszej do największej. Do przechowywania wczytywanych liczb wykorzystaj szablon `tablica` z zadania 5.43.
Rozwiązanie zadania 5.43 umieść w oddzielnym module znajdującym się w oddzielnym pliku.

5.49 Napisz program wczytujący ze standardowego wejścia i przechowujący podaną przez użytkownika liczbę wartości typu `napis` z zadania 5.44. Do przechowywania napisów wykorzystaj szablon `tablica` z zadania 5.43. Rozwiązania zadań 5.43 oraz 5.44 i 5.45 umieść w dwóch osobnych modułach znajdujących się w oddzielnych plikach.

5.50 Napisz program wczytujący ze standardowego wejścia pewną podaną przez użytkownika liczbę wartości typu `int` i wypisujący je na standardowym wyjściu w odwrotnej kolejności niż kolejność wczytania. W rozwiązaniu wykorzystaj rozwiązania zadań 5.38, 5.39 i 5.41 . Kod programu rozmieść w kilku plikach. Stwórz trzy moduły: pierwszy zawierający rozwiązanie zadania 5.38, drugi z rozwiązaniem zadania 5.39 oraz trzeci z rozwiązaniem zadań 5.41.

Rozdział 6

STL

W niniejszym rozdziale znajdują się zadania pozwalające przećwiczyć wykorzystanie różnych elementów biblioteki STL. Podobne zadania dotyczące operacji na plikach oraz na napisach czytelnik znajdzie w pierwszej części skryptu.

6.1 **(r)** Napisz program, który wczytuje ze standardowego wejścia dodatnie liczby całkowite. Program powinien skończyć wczytywanie liczb po wczytaniu liczby 0. Na koniec program powinien wypisać na standardowym wyjściu wszystkie wczytane liczby łącznie z zerem.
W rozwiązaniu zadania wykorzystaj któryś z zdefiniowanych w STL-u kontenerów.

6.2 Napisz program, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą `n`, następnie `n` liczb całkowitych i wypisuje na standardowym wyjściu wczytane liczby.
W rozwiązaniu zadania wykorzystaj któryś z zdefiniowanych w STL-u kontenerów.

6.3 **(r,!)** Napisz funkcję, która otrzymuje jako argument referencję do obiektu klasy `vector<int>` i:
   a) odwraca kolejność elementów w otrzymanym w argumencie kontenerze,
   b) sortuje rosnąco elementy otrzymanego w argumencie kontenera,
   c) sortuje malejąco elementy otrzymanego w argumencie kontenera,
   d) sortuje rosnąco elementy otrzymanego w argumencie kontenera względem ich wartości bezwzględnych,
   e) sortuje rosnąco elementy otrzymanego w argumencie kontenera względem ich reszty z dzielenia przez 1000.

6.4 **(r,C++11)** Jak różniłoby się rozwiązanie zadania 6.3, gdyby argumentem funkcji była referencja do obiektu klasy `array<int,n>`, gdzie `n` jest pewną liczbą całkowitą. Czy można zaimplementować rozwiązanie tak, żeby działało dla obiektów klasy `array<int,n>` o dowolnym `n`?

6.5 **(r)** Napisz klasę `macierz`, która służy do przechowywania macierzy liczb wymiernych. Klasa `macierz` powinna udostępniać następujące publiczne metody:
   — konstruktor, który otrzymuje dwie liczby całkowite i inicjuje macierz o wymiarach podanych w argumentach,
   — przeciążony operator `()`, który dla liczb całkowitych i, j podanych jako argumenty zwraca referencję do elementu macierzy znajdującego się i-tej kolumnie i j-wierszu.

   Do implementacji klasy `macierz` wykorzystaj typ `vector`.

6.6 **(C++11)** Napisz klasę `macierz1010` służącą do przechowywania macierzy liczb wymiernych o wymiarach 10 na 10. Klasa `macierz1010` powinna udostępniać przeciążony operator `()`, który dla liczb całkowitych

i, j podanych jako argumenty zwraca referencję do elementu macierzy znajdującego się i-tej kolumnie i j-wierszu.
Do implementacji klasy `macierz` wykorzystaj typ `array`.

6.7 Napisz klasę `tablica3D` analogiczną do klasy `macierz` z zadania 6.5 ale przechowującą trójwymiarową tablicę przy użyciu obiektów typu `vector`.

6.8 Napisz program, który wczytuje ze standardowego wejścia dodatnie liczby całkowite n i m ($m \leq n$) oraz n liczb całkowitych, a następnie wypisuje na standardowym wyjściu m-tą pod względem wielkości liczbę.

6.9 **(r)** Napisz program, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n a następnie wczytuje n par liczb całkowitych i wstawia je do listy (liczby nieujemne program powinien wstawiać na koniec listy, a ujemne na początek listy). Na końcu program powinien wypisać na standardowym wyjściu wszystkie wczytane elementy w kolejności ich występowania na liście.
W rozwiązaniu wykorzystaj kontener `list`.

6.10 Zaimplementuj klasę `kolejka` z zadania 1.46 przy użyciu kontenera `list`.

6.11 Napisz klasę służącą do przechowywania listy napisów. Klasa ta powinna udostępniać następujące publiczne metody:
— `wczytaj` wczytującą ze standardowego wejścia napis i umieszczającą go na końcu przechowywanej listy,
— `wypisz` wypisującą na standardowym wyjściu dziesięć pierwszych elementów listy. Metoda `wypisz` powinna usunąć z listy wypisane elementy.
W rozwiązaniu zadania użyj szablonu `queue`.

6.12 **(r)** W rozwiązaniu zadania 6.11 użyj szablonu klasy `queue` w taki sposób, żeby używał kontenera `list` w miejsce `dequeue`.

6.13 **(*)** Napisz program wczytujący ze standardowego wejścia wyrażenie arytmetyczne zapisane w odwrotnej notacji polskiej i wypisujący na standardowym wyjściu jego wartość. Użyj w programie szablonu `stack` z STL-a.

6.14 Napisz klasę `bufor` udostępniającą publiczną metodę `dodaj` dodającą do bufora napis podany w argumencie metody. W przypadku dodania do bufora dziesiątego napisu metoda `dodaj` powinna wypisać na standardowym wyjściu przechowywane w buforze napisy w kolejności ich dodawania i opróżnić bufor.

6.15 Klasę `bufor` z zadania 6.14 napisz w taki sposób, żeby funkcja bufor zamiast na standardowe wyjście zapisywała przechowywane napisy do strumienia podanego w argumencie konstruktora.

6.16 Napisz funkcję, która dostaje jako argument napis zawierający wyrażenie arytmetyczne zapisane w odwrotnej notacji polskiej i zwraca wartość

wyrażenia otrzymanego w argumencie. Zakładamy, że liczby oraz operatory arytmetyczne w wyrażeniu są oddzielone pojedynczymi znakami odstępu. Do manipulowania na napisie użyj klasy `stringstream`.

6.17 Napisz program, który wczytuje ze standardowego wejścia nazwę pliku, dodatnią liczbę całkowitą n oraz n liczb całkowitych, a następnie zapisuje w sposób tekstowy przeczytane liczby do pliku o podanej nazwie w kolejności od najmniejszej do największej.

6.18 Napisz program, który wczytuje ze standardowego wejścia nazwę pliku tekstowego, czyta zawartość pliku (zakładamy, że w pliku znajdują się liczby oddzielone białymi znakami oraz że cała zawartość pliku zmieści się w pamięci programu) i wypisuje na standardowym wyjściu medianę z liczb przeczytanych z pliku.

6.19 **(r)** Napisz klasę `rekrutacja` służącą do przechowywania i wyszukiwania danych kandydatów na pracowników. Klasa `rekrutacja` powinna przechowywać następujące dane kandydatów: imię, nazwisko, pesel, telefon kontaktowy oraz liczbę punktów zdobytych podczas rekrutacji i udostępniać następujące publiczne metody:
- `dodaj` dodającą do listy kandydata o danych podanych jako argumenty metody,
- `najlepszy` zwracającą jako wartość strukturę zawierającą dane kandydata o maksymalnej liczbie punktów zdobytych podczas rekrutacji spośród kandydatów przechowywanych w bazie. Jeżeli więcej niż jeden kandydat ma maksymalną liczbę punktów metoda powinna zwrócić któregokolwiek z nich.
- `usun` usuwającą z bazy dane kandydata o największej liczbie punktów zdobytych podczas rekrutacji. W przypadku gdy więcej niż jeden kandydat ma maksymalną liczbę punktów metoda powinna usunąć tego z nich, którego zwróciłaby metoda `najlepszy`.

W rozwiązaniu użyj szablonu `priority_queue`.

6.20 Napisz klasę `rekrutacja` w taki sposób, żeby `priority_queue` używała `dequeue`.

6.21 Napisz funkcję sortującą elementy otrzymanej w argumencie listy liczb całkowitych przy wykorzystaniu szablonu klasy `priority_queue`.

6.22 **(*)** Zaimplementuj klasę `rekrutacja` z zadania 6.19 używając wektora oraz operacji `make_heap`, `push_heap` oraz `pop_heap` zamiast klasy `priority_queue`.

6.23 Napisz klasę `punkty` przechowującą współrzędne punktów w trójwymiarowym kartezjańskim układzie współrzędnych udostępniającą następujące publiczne metody:
- `dodaj` dodającą do przechowywanych punktów punkt o współrzędnych podanych w argumencie funkcji,

— `najblizszy` podającą współrzędne przechowywanego punktu najbliższego początkowi układu współrzędnych.

6.24 **(r)** Napisz klasę `slownik` służącą do przechowywania słownika ortograficznego. Klasa `slownik` powinna udostępniać następujące publiczne metody:
— `dodaj` - dodającą do słownika wyraz podany w argumencie metody,
— `znajdz` - sprawdzającą czy podany w argumencie wyraz należy do słownika (czyli czy jest poprawnie zapisanym słowem). Metoda powinna zwrócić `true` jeżeli podany w argumencie wyraz należy do słownika i `false` w przeciwnym wypadku.

6.25 **(r)** Do klasy `slownik` z zadania 6.24 dopisz metodę `zakres`, która dla dwóch słów `slowo1` i `slowo2` podanych w argumentach wypisuje na standardowym wyjściu wszystkie słowa należące do słownika znajdujące się w porządku alfabetycznym po `slowo1` a przed `slowo2`.

6.26 Napisz funkcję `suma`, która otrzymuje jako argumenty stałe referencje do dwóch zbiorów (typ `set`) o elementach typu `int` i zwraca jako wartość zbiór zawierający teoriomnogościową sumę zbiorów otrzymanych w argumentach.

6.27 Napisz funkcję `suma` analogiczną do tej z zadania 26, ale działającą na multizbiorach. Zgodnie z definicją sumy na multizbiorach liczba wystąpień poszczególnych elementów w wynikowym multizbiorze powinna być sumą liczby wystąpień tych elementów w multizbiorach będących argumentami sumy.

6.28 Napisz klasę `encyklop` służącą do przechowywania haseł encyklopedii. Klasa `encyklop` powinna udostępniać następujące publiczne metody:
— `dodaj`, która dodaje do encyklopedii hasło i jego definicję podane jako argumenty,
— `znajdz`, która zwracaj jako wartość definicję hasła podanego w argumencie. Jeżeli dane hasło nie należy do encyklopedii metoda powinna zwrócić pusty napis (zakładamy, że hasło może mieć co najwyżej jedną definicję).

6.29 Przerób klasę `encyklop` z zadania 6.28 w taki sposób, żeby umożliwiały przechowywanie wielu definicji dla jednego hasła. Metoda `znajdz` powinna zwracać jako wartość listę definicji dla podanego w argumencie hasła.

6.30 Do klas `encyklop` z zadań 6.28 i 6.29 dodaj publiczną metodę `zakres` wypisującą na standardowym wyjściu definicje haseł z zakresu podanego w argumentach metody (dla argumentów `haslo1` i `haslo2` metoda powinna wypisać definicje haseł następujących w kolejności leksykograficznej po `haslo1` a przed `haslo2`).

6.31 Napisz klasę `uczniowie` służącą do przechowywania adresów uczniów. Klasa `uczniowie` powinna udostępniać następujące publiczne metody:

— `dodaj` dodającą ucznia, którego dane zostały podane w argumentach,
— `adres` zwracająca jako wartość adres ucznia o imieniu i nazwisku podanych w argumentach,
— `lista` wypisującą na standardowym wyjściu przechowywane dane uczniów. Dane powinny być wypisywane w porządku alfabetycznym względem nazwisk i imion uczniów.

Implementując klasę `uczniowie` dopuść możliwość, że dwaj różni uczniowie mogą mieć takie samo imię i nazwisko.

6.32 **(r)** Napisz szablon funkcji, która dostaje jako argumenty iteratory (forward iterators) do początku oraz końca przedziału kontenera i wypisuje na standardowym wyjściu wszystkie elementy przedziału.
Pamiętaj, że w STL-u przedział wyznaczony przez iteratory `poczatek` i `koniec` zawiera `*poczatek`, a nie zawiera `*koniec`. Iterator `koniec` wskazuje tuż za koniec przedziału.

6.33 Napisz szablon funkcji, która dostaje jako argumenty iteratory (forward iterators) do początku oraz końca przedziału kontenera (zakładamy, że elementami przedziału są liczby) i wypełnia otrzymany w argumencie przedział zerami.

6.34 Napisz szablon funkcji, która dostaje jako argumenty iteratory (forward iterators) do początku oraz końca przedziału kontenera (zakładamy, że elementami przedziału są liczby) i zwraca jako wartość drugi co do wielkości element przedziału otrzymanego w argumentach.

6.35 Napisz szablon funkcji, która dostaje jako argumenty iteratory (bidirectorial iterators) do początku oraz końca przedziału kontenera i odwraca kolejność elementów przedziału. Rozwiąż zadanie nie korzystając z biblioteki `algorithm`.

6.36 Napisz szablon funkcji, która otrzymuje jako argumenty iteratory (random access iterators) do początku oraz końca przedziału kontenera i zwraca jako wartość losowy element otrzymanego w argumentach przedziału.

6.37 Napisz szablon funkcji, która dostaje jako argumenty iteratory (random access iterators) do początku oraz końca przedziału kontenera i w losowy sposób zmienia kolejność elementów otrzymanego w argumencie przedziału.

6.38 **(r)** Napisz funkcję `vec_abs`, która otrzymuje jako argument referencję do wektora o elementach typu `int` i nadaje wszystkim elementom wektora ich wartość bezwzględne. W funkcji wykorzystaj szablon `for_each` z biblioteki `algorithm`.

6.39 **(r,C++11)** Napisz funkcję, która dostaje jako argument wektor o elementach typu `int` i zwraca `true` jeżeli

a) któryś z elementów wektora jest podzielny przez liczbę z zakresu od 2 do 10,
b) żaden z elementów wektora nie jest podzielny przez liczbę z zakresu od 2 do 10
c) każdy z elementów jest podzielny przez liczbę z zakresu od 2 do 10

Do rozwiązanie zadania wykorzystaj, któryś z szablonów `all_of`, `any_of`, `none_of` z biblioteki `algorithm`.

6.40 Napisz funkcję, która dostaje jako argument wektor o elementach typu `int` i zwraca:
a) liczbę elementów wektora równych 0,
b) liczbę elementów wektora z przedziału od 2 do 10
c) liczbę elementów parzystych wektora.

Do rozwiązanie zadania wykorzystaj któryś z szablonów `count`,`count_if` z biblioteki `algorithm`.

6.41 Napisz funkcję, która dostaje jako argumenty dwa wektory `w1`, `w2` liczb całkowitych i:
a) wypełnia zerami pozycje, na których wektory `w1` i `w2` się różnią (jeżeli `w1[i]!=w2[i]`, to funkcja ma nadać elementom `w1[i]` i `w2[i]` wartość `0`),
b) wypełnia zerami pozycje, na których wektory `w1` i `w2` mają elementy dające różną resztę z dzielenia przez `2`.

Do rozwiązanie zadania wykorzystaj szablon `mismatch` z biblioteki `algorithm`.

6.42 **(r)** Napisz funkcję, która otrzymuje jako argument napis typu `string` i zwraca wartość `true` wtedy i tylko wtedy, gdy podany w argumencie napis jest palindromem. W rozwiązaniu zadania wykorzystaj szablon `equal` z biblioteki `algorithm`.

6.43 Napisz funkcję, która otrzymuje jako argument wektor `wek` liczb całkowity i zwraca jako wartość:
a) indeks pierwszego elementu równego 0 wektora `wek` lub -1 jeżeli taki element nie istnieje,
b) wartość pierwszego dodatniego elementu wektora `wek` lub 0 jeżeli taki element nie istnieje.

W funkcji wykorzystaj któryś z szablonów `find`, `find_if`, `find_if_not` (ten ostatni tylko w C++11) z biblioteki `algorithm`.

6.44 Napisz funkcję, która otrzymuje jako argument wektor `wek` dodatnich liczb całkowitych i:
a) zwraca jako wartość indeks pierwszego wystąpienia w wektorze `wek` liczby pierwszej nie większej niż 100,
b) zwraca jako wartość indeks pierwszego wystąpienia w wektorze `wek` liczby podzielnej przez liczbę pierwszą nie większą niż 100.
c) wypisuje na standardowym wyjściu wszystkie występujące w wek-

torze `wek` liczby podzielne prze jakąś liczbę pierwszą mniejszą niż 100.

Do rozwiązanie zadania wykorzystaj szablon `find_first_of` z biblioteki `algorithm`.

6.45 **(C++11)** Napisz funkcję, która dostaje dwa argumenty stałą referencje `wek1` do wektora liczb całkowitych oraz referencję `wek2` do wektora liczb całkowitych i przekopiowuje do `wek2` wszystkie parzyste liczby zawarte w `wek1`. Do rozwiązania wykorzystaj szablon `copy_if` z biblioteki `algorithm`.

6.46 Napisz funkcję, która sortuje wektor liczb wymiernych otrzymanych w argumencie przy pomocy algorytmu quicksort. W funkcji wykorzystaj szablon `partition` z biblioteki `algorithm`.

6.47 **(**)** Napisz własny alokator, który posiada dużą statyczną automatyczną tablicę o elementach będących typu będącego parametrem szablonu i udostępnia jej fragmenty. Napisz alokator tak, aby mógł być używany jako parametr w kontenerach STL-a (np.: jako drugi parametr szablonu klasy `vector`).

Rozdział 7

Wyjątki

7.1 **(r)** Napisz funkcję, która otrzymuje jako argument dodatnią liczbę `n`, alokuje `n`-elementową tablicę o elementach typu `double` i zwraca jako wartość wskaźnik do pierwszego elementu świeżo utworzonej tablicy.
W przypadku gdy alokacja tablicy się nie powiedzie, funkcja powinna wypisać na standardowym wyjściu komunikat:
`Nieudana proba alokacji tablicy` i zwrócić jako wartość `NULL`.
Zadanie rozwiąż na dwa sposoby, używając dwóch wersji operatora `new`: rzucającej i nierzucającej wyjątki.

7.2 **(r)** Napisz funkcję `zamien`, która otrzymuje jako argument trzy wektory `v1`, `v2`,`v3` liczb całkowitych oraz trzy nieujemne liczby całkowite `l1`, `l2`, `l3` i zamienia ze sobą wartościami elementy `v1[l1]`, `v2[l2]` oraz `v3[l3]` w taki sposób, że stara wartość `v1[l1]` ma być zapisana w `v2[l2]`, stara wartość `v2[l2]` w `v3[l3]`, zaś stara wartość `v3[l3]` w `v1[l1]`. W przypadku gdy któryś z indeksów `l1`, `l2`, `l3` leży poza zakresem indeksów wektora, do którego się odnosi, funkcja powinna rzucić standardowy wyjątek `std::out_of_rang`. Przed rzuceniem wyjątku funkcja nie powinna zmieniać wartości żadnego z elementów wektorów `v1`, `v2`,`v3`.

7.3 Operator `[]` w klasie `tablica` z zadania 4.4 przeciąż w taki sposób, żeby przy podaniu indeksu spoza zakresu dozwolonych indeksów rzucał wyjątek `std::out_of_range`.

7.4 **(r)** Zdefiniuj hierarchię wyjątków występujących przy operacjach na zmiennych liczbowych przechowujących liczby wymierne. Uwzględnij wyjątki rzucane w przypadku: dzielenia przez zero, liczenia pierwiastka kwadratowego z liczby ujemnej, użycia w działaniu niezainicjowanej zmiennej, próbie przypisania wartości do stałej, działania o wyniku spoza zakresu itd.

7.5 **(rozw)** Napisz funkcję będącą obudową funkcji
a) `pow`
b) `sqrt`
z biblioteki `cmath`. Twoja funkcja powinna różnić się od oryginalnej funkcji tym, że zamiast zmieniać wartość makra `errno` w przypadku błędu, rzuca jeden z wyjątków z hierarchii zdefiniowanej w zadaniu 7.4.

7.6 **(r,C)** Napisz funkcje obudowujące działania arytmetyczne na zmiennych typu `double` w taki sposób, że w przypadku wystąpienia błędu funkcje te:
a) ustawiają odpowiednią wartość błędu w makrze `errno`,
b) rzucają odpowiedni wyjątek z hierarchii wyjątków zdefiniowanej w zadaniu 7.4.

7.7 **(r,C)** Napisz funkcję `rownanie`, która otrzymuje jako argumenty zmienne `a`, `b`, `c` typu `double` i wypisuje na standardowym wyjściu rozwiązania równania $a \cdot x^2 + b \cdot x + c = 0$ lub napis `nie ma rozwiazania`. Funk-

cję napisz tak, żeby wykrywała sytuacje, w których rozwiązanie może nie być poprawne. W funkcji wykorzystaj rozwiązania zadań 7.5 i 7.6. Porównaj rozwiązania bazujące na wyjątkach oraz na makrze `errno`.

7.8 **(r)** Napisz wersję funkcji `rownanie` z zadania 7.7, która po złapaniu jakiegoś wyjątku wypisuje odpowiedni komunikat na standardowym wyjściu i rzuca z powrotem wyjątek.

7.9 Napisz klasę `nowy_double` służącą do przechowywania zmiennych typu `double`. Obiekty klasy `nowy_double` powinny pamiętać, czy zostały zainicjowane jakąś wartością, czy nie, i czy są stałe, czy nie. Klasa `nowy_double` powinna posiadać:
- bezargumentowy konstruktor inicjujący obiekt o niezdefiniowanej przechowywanej wartości,
- jednoargumentowy konstruktor inicjujący obiekt o wartości podanej w argumencie konstruktora,
- konstruktor kopiujący,
- przeciążone operatory arytmetyczne w taki sposób, aby w przypadku niedozwolonego działania (na przykład dzielenia przez 0 lub podania jako argumentu zmiennej o nieokreślonej wartości) rzucały odpowiedni wyjątek z hierarchii wyjątków zdefiniowanej w zadaniu 7.4,
- bezargumentowe metody `stala`, `zmienna` powodujące ustalenie wartości przechowywanej w obiekcie jako odpowiednio stałej lub zmiennej.
- operator przypisania, który rzuca odpowiedni wyjątek z hierarchii zdefiniowanej w zadaniu 7.4 w przypadku użycia po lewej stronie operatora obiektu przechowującego wartość oznaczoną jako stałą.

7.10 Napisz funkcję zwracającą jako wartość pierwiastek kwadratowy z liczby typu `nowy_double` otrzymanej w argumencie. Zwrócona wartość powinna być typu `nowy_double`. W przypadku gdy argument jest liczbą ujemną lub ma nieokreślona wartość funkcja, powinna rzucić odpowiedni wyjątek z hierarchii zdefiniowanej w zadaniu 7.4.

7.11 Napisz funkcję, która otrzymuje jako argumenty zmienne `a`, `b`, `c` typu `nowy_double` z zadania 7.9 i wypisuje na standardowym wyjściu rozwiązania równania $a \cdot x^2 + b \cdot x + c = 0$ lub napis `nie ma rozwiazania`. Zadanie rozwiąż na dwa sposoby z wykorzystaniem instrukcji `if` oraz bez instrukcji `if`, ale przy użyciu wyjątków.

7.12 Stwórz hierarchię wyjątków służącą do obsługi zdarzeń związanych z alokacją i obsługą tablicy. Uwzględnij takie zdarzenia jak: problem z alokacją tablicy, odnoszenie się do elementu tablicy za pomocą indeksu mniejszego niż 0, odnoszenie się do elementu tablicy za pomocą zbyt dużego indeksu, próba dereferencji iteratora nie zainicjowanego żadną wartością, próba dereferencji iteratora wskazującego poza zakres tablicy,

próba dereferencji nieaktualnego iteratora, próba inkrementacji/dekrementacji iteratora nie zainicjowanego żadną wartością itd.

7.13 Napisz klasę `tablica` służącą do przechowywania tablicy liczb całkowitych. Klasa `tablica` powinna udostępniać:
- konstruktor tworzący tablicę liczb całkowitych o rozmiarze podanych w argumencie, w przypadku problemów z alokacją tablicy konstruktor powinien rzucić odpowiedni wyjątek,
- przeciążony operator `[]`, w przypadku podania niewłaściwego indeksu powinien rzucić odpowiedni wyjątek.

W zadaniu wykorzystaj wyjątki zdefiniowane w zadaniu 7.12.

7.14 Zdefiniuj iterator dla klasy `tablica` z zadania 7.13. Iterator powinien mieć przeciążone operatory `++`, `--`, `*` (operator dereferencji). W przypadku niemożności wykonania odpowiedniej operacji (na przykład operacji arytmetycznej na niezainicjowanym iteratorze, albo dereferencji nieaktualnego lub niewskazującego na żaden element tablicy iteratora) operator powinien rzucić odpowiedni wyjątek zdefiniowany w zadaniu 7.12.

7.15 Napisz funkcję, która otrzymuje jako argumenty dwie referencje do iteratorów o typie zdefinowanym w zadaniu 7.14 wskazujące na początek i koniec pewnego zakresu, i wypisuje na standardowym wyjściu wszystkie wartości z podanego zakresu. Funkcja powinna obsługiwać wszystkie mogące wystąpić wyjątki i informować o nich odpowiednim komunikatem wypisanym na standardowym wyjściu.

7.16 Napisz wersję funkcji z zadania 7.15, która w przypadku złapania jakiegoś wyjątku poza wypisaniem odpowiedniego komunikatu na standardowym wyjściu rzuca dalej ten sam wyjątek.

7.17 **(r)** Napisz funkcję `bezpieczna`, która wypisuje na standardowym wyjściu `jestem bezpieczna`. Funkcję `bezpieczna` zadeklaruj jako nierzucającą wyjątków.

7.18 **(r)** Napisz funkcję `podziel`, która dla dwóch argumentów całkowitoliczbowych zwraca jako wartość ich iloraz. W przypadku dzielenia przez zero funkcja powinna rzucić odpowiedni wyjątek. W deklaracji funkcji zdefiniuj wyjątki jakie może rzucić ta funkcja.

7.19 **(r,!,*)** Napisz program, który wczytuje ze standardowego wejścia 10 par liczb i wypisuje na standardowym wyjściu ilorazy wszystkich tych par, w których druga liczba jest różna od 0. Do dzielenia wykorzystaj funkcję `podziel` z zadania 7.18. Przechwytuj wyjątki rzucane przez funkcję `podziel`. Zdefiniuj swoją funkcję `unexpected` w taki sposób, żeby w wypadku rzucenia w funkcji `podziel` wyjątku spoza listy wypisała ona komunikat `nieznany blad przy dzieleniu` i rzuciła dalej zdefiniowany na taką okoliczność wyjątek. Uwzględnij ten wyjątek w liście dopuszczalnych wyjątków funkcji `podziel`.

7.20 **(r,!,*)** W programie z zadania 7.19 zdefiniuj swoją funkcję `terminate` wypisującą na standardowym wyjściu komunikat `niezlapany wyjatek` i wywołującą funkcje `exit`.

7.21 Zadania od 7.9 do 7.11 oraz od 7.13 do 7.16 rozwiąż w taki sposób, żeby wszystkie zaimplementowane funkcje i metody deklarowały listę wyjątków jakie mogą rzucić.

7.22 **(r,C,!,*)** Zadania 7.18 i 7.19 napisz symulując wyjątki przy pomocy operacji `setjmp` i `longjmp`. W nowym rozwiązaniu zadania 7.19 nie definiuj swojej funkcji `unexpected`.

Rozdział 8

Rozwiązania

8.1. Rozwiązania zadań z rozdziału 1

Zadanie 1.1

Listing 8.1. rozwiązanie zadania 1.1

```
class poczta {
  public:
      std::string nadawca, odbiorca, temat, tresc;
};
```

Zadanie 1.2

Listing 8.2. rozwiązanie zadania 1.2

```
void wypisz(poczta p){
    std::cout<<"nadawca : "<<p.nadawca<<std::endl;
    std::cout<<"odbiorca : "<<p.odbiorca<<std::endl;
    std::cout<<"temat : "<<p.temat<<std::endl;
    std::cout<<"tresc : "<<p.tresc<<std::endl;
}
```

Zadanie 1.3

Listing 8.3. rozwiązanie zadania 1.3

```
void wczytaj(poczta &p){
    std::cout<<"Podaj nadawce : ";
    std::cin>>p.nadawca;
    std::cout<<"Podaj odbiorce : ";
    std::cin>>p.odbiorca;
    std::cin.ignore();
    std::cout<<"Podaj temat : ";
    getline(std::cin,p.temat);
    std::cout<<"Podaj tresc : ";
    getline(std::cin,p.tresc);
}
```

Zadanie 1.4

Listing 8.4. rozwiązanie zadania 1.4

```
class poczta {
public:
    std::string nadawca, odbiorca, temat, tresc;

    void wczytaj();
    void wypisz();
};
```

```
9  void poczta::wypisz(){
       std::cout<<"nadawca : "<<nadawca<<std::endl;
11     std::cout<<"odbiorca : "<<odbiorca<<std::endl;
       std::cout<<"temat : "<<temat<<std::endl;
13     std::cout<<"tresc : "<<tresc<<std::endl;
   }
15

17 void poczta::wczytaj(){
       std::cout<<"Podaj nadawce : ";
19     std::cin>>nadawca;
       std::cout<<"Podaj odbiorce : ";
21     std::cin>>odbiorca;
       std::cin.ignore();
23     std::cout<<"Podaj temat : ";
       getline(std::cin,temat);
25     std::cout<<"Podaj tresc : ";
       getline(std::cin,tresc);
27 }
```

Zadanie 1.5 Struktury i klasy to w języku C++ pojęcia niemal tożsame. Różnią się szczegółami takimi jak to, że pola struktur domyślnie są publiczne, zaś pola klas domyślnie są prywatne. Poniżej definicja struktury `poczta2` oraz operujących na niej funkcji:

Listing 8.5. rozwiązanie zadania 1.5

```
1  struct poczta2 {
       std::string nadawca, odbiorca, temat, tresc;
3  };

5  void wypisz(poczta2 p){
       std::cout<<"nadawca : "<<p.nadawca<<std::endl;
7      std::cout<<"odbiorca : "<<p.odbiorca<<std::endl;
       std::cout<<"temat : "<<p.temat<<std::endl;
9      std::cout<<"tresc : "<<p.tresc<<std::endl;
   }
11
   void wczytaj(poczta2 &p){
13     std::cout<<"Podaj nadawce : ";
       std::cin>>p.nadawca;
15     std::cout<<"Podaj odbiorce : ";
       std::cin>>p.odbiorca;
17     std::cin.ignore();
       std::cout<<"Podaj temat : ";
19     getline(std::cin,p.temat);
       std::cout<<"Podaj tresc : ";
21     getline(std::cin,p.tresc);
   }
```

Poniżej wersja struktury `poczta2` z dopisanym metodami do wczytywania i wypisywania zawartości:

Listing 8.6. rozwiązanie zadania 1.5

```
struct poczta2 {
    std::string nadawca, odbiorca, temat, tresc;

    void wczytaj();
    void wypisz();
};


void poczta2::wypisz(){
    std::cout<<"nadawca : "<<nadawca<<std::endl;
    std::cout<<"odbiorca : "<<odbiorca<<std::endl;
    std::cout<<"temat : "<<temat<<std::endl;
    std::cout<<"tresc : "<<tresc<<std::endl;

}


void poczta2::wczytaj(){
    std::cout<<"Podaj nadawce : ";
    std::cin>>nadawca;
    std::cout<<"Podaj odbiorce : ";
    std::cin>>odbiorca;
    std::cout<<"Podaj temat : ";
    std::cin.ignore();
    getline(std::cin,temat);
    std::cout<<"Podaj tresc : ";
    getline(std::cin,tresc);
}
```

Zadanie 1.15

Listing 8.7. rozwiązanie zadania 1.15

```
class liczba{
private:
    int wart_licz;
public:
    void wczytaj();
    void wypisz();
    void nadaj_w(int);
    int wartosc();
    unsigned int abs();
};

void liczba::wczytaj(){
    std::cout<<"Podaj wartosc liczby : ";
```

```
14      std::cin>>wart_licz;
    }
16
    void liczba::wypisz(){
18      std::cout<<"Przechowywana liczba to : "<<wart_licz<<std::endl;
    }
20
    void liczba::nadaj_w(int n){
22      wart_licz=n;
    }
24
    int liczba::wartosc(){
26      return wart_licz;
    }
28
    unsigned int liczba::abs(){
30      if (wart_licz>=0)
            return wart_licz;
32      else
            return -wart_licz;
34  }
```

Zadania 1.17–1.20

Listing 8.8. rozwiązanie zadań 1.17–1.20

```
    class punkt{
2       int x,y;
        friend punkt rzutuj(punkt3);
4   public:
        void rzutuj(punkt3);
6       void wczytaj();
        void wypisz();
8   };

10  class punkt3{
        int x,y,z;
12      friend punkt rzutuj(punkt3);
        friend void punkt::rzutuj(punkt3);
14  public:
        void wczytaj();
16      void wypisz();
    };
18
    void punkt::wczytaj(){
20      std::cout<<"Podaj wspolrzedne punktu : ";
        std::cin>>x>>y;
22  }

24  void punkt::wypisz(){
        std::cout<<"Wspolrzedne punktu to : ";
```

```
26      std::cout<<x<<" "<<y<<std::endl;
    }
28

30  void punkt3::wczytaj(){
        std::cout<<"Podaj wspolrzedne punktu :";
32      std::cin>>x>>y>>z;
    }
34
    void punkt3::wypisz(){
36      std::cout<<"Wspolrzedne punktu to:";
        std::cout<<x<<" "<<y<<" "<<z<<std::endl;
38  }
    punkt rzutuj(punkt3 p3){
40      punkt p2;
        p2.x=p3.x;
42      p2.y=p3.y;
        return p2;
44  }
    void punkt::rzutuj(punkt3 p3){
46      x=p3.x;
        y=p3.y;
48  }
```

Zadanie 1.26

Listing 8.9. rozwiązanie zadania 1.26

```
    class ukryta_liczba{
2   private:
        int liczba;
4   public:
        void zeruj();
6   };
    void ukryta_liczba::zeruj(){
8       liczba=0;
    }
```

Zadanie 1.27

Listing 8.10. rozwiązanie zadania 1.27

```
1   class ukryta_liczba{
    private:
3       int liczba;
        friend void inkrementuj(ukryta_liczba&);
5   public:
        void zeruj();
7   };
```

```
9  void ukryta_liczba::zeruj(){
       liczba=0;
11 }

13 void inkrementuj(ukryta_liczba& u){
       u.liczba++;
15 }
```

Zadanie 1.28

Listing 8.11. rozwiązanie zadania 1.28

```
   class ukryta_liczba{
2  private:
       int liczba;
4  public:
       void zeruj();
6      void inkrementuj(ukryta_liczba&);
   };
8
   void ukryta_liczba::zeruj(){
10     liczba=0;
   }
12
   void ukryta_liczba::inkrementuj(ukryta_liczba& u){
14     u.liczba++;
   }
```

Zadania 1.33-1.35

Listing 8.12. rozwiązanie zadań 1.33–1.35

```
1
   class wskaznik{
3  private:
       int *wsk;
5  public:
       wskaznik();
7      void utworz(int);
       int* zwroc();
9      void zwolnij();
       void kopiuj(wskaznik&);
11     ~wskaznik();
       friend void przepisz(int*,wskaznik &);
13 };

15
   wskaznik::wskaznik(){
17     wsk=NULL;
   }
```

```
19
   void wskaznik::utworz(int n){
21     wsk = new int[n];
   }
23
   int* wskaznik::zwroc(){
25     return wsk;
   }
27
   void wskaznik::zwolnij(){
29     delete [] wsk;
       wsk = NULL;
31 }

33 void wskaznik::kopiuj(wskaznik & ref){
       ref.wsk=wsk;
35 }

37 wskaznik::~wskaznik(){
       if (wsk!=NULL)
39         delete [] wsk;
   }
41
   void przepisz(int * t, wskaznik& ref){
43     ref.wsk = t;
   }
```

Zadanie 1.36

Listing 8.13. rozwiązanie zadania 1.36

```
   class identyfikator {
2  private:
       unsigned int liczba;
4  public:
       unsigned int id();
6  };

8  unsigned int identyfikator::id(){
       return liczba++;
10 }
```

Zadanie 1.37

Listing 8.14. rozwiązanie zadania 1.37

```
   class identyfikator2 {
2  private:
       unsigned int liczba;
4
```

```
public:
    identyfikator2();
    unsigned int id();
};

identyfikator2::identyfikator2(){
    liczba=0;
}

unsigned int identyfikator2::id(){
    return liczba++;
}
```

Zadanie 1.38

Listing 8.15. rozwiązanie zadania 1.38

```
class semafor_bin{
private:
    bool wolny;
public:
    semafor_bin(){wolny=true;}
    void rezerwuj(){wolny=false;}
    void zwolnij(){wolny=true;}
    bool stan(){return wolny}
};
```

Zadanie 1.43

Listing 8.16. rozwiązanie zadania 1.43

```
class  osoba{
private:
    std::string imie, nazwisko;
public:
    osoba(std::string, std::string);
    void wczytaj();
    void wypisz();
};

osoba::osoba(std::string im, std::string nazw){
    imie=im;
    nazwisko=nazw;
}
void osoba::wczytaj(){
    std::cout<<"Podaj imie : ";
    std::cin>>imie;
    std::cout<<"Podaj nazwisko : ";
    std::cin>>nazwisko;
}
```

```
void osoba::wypisz(){
    std::cout<<"imie"<<imie<<std::endl;
    sts::cout<<"nazwisko"<<nazwisko<<std::endl;
}
```

Zadanie 1.44

Listing 8.17. rozwiązanie zadania 1.44

```
std::vector<osoba> ZrobWektor(int n){
    return  std::vector<osoba>(n,osoba("Jan", "Kowalski"));
}
```

Zwróćmy uwagę, że nie moglibyśmy napisać podobnej funkcji zwracającej jako wartość tablicy obiektów klasy `osoba`. Jest to spowodowane tym, że klasa `osoba` nie posiada bezargumentowego konstruktora. Pewne rozwiązanie tego problemu można znaleźć w rozwiązaniu kolejnego zadania.

Zadanie 1.45

Listing 8.18. rozwiązanie zadania 1.45

```
class tab_osoba{
private:
    osoba ** tab;
    int rozmiar;
public:
    tab_osoba(int, std::string, std::string);
    osoba& at(int);
    ~tab_osoba();
};

tab_osoba::tab_osoba(int n, std::string imie,
                                    std::string nazwisko){
    tab = new osoba*[n];
    rozmiar=n;
    for(int i=0;i<n;i++)
        tab[i] = new osoba(imie,nazwisko);
}

osoba& tab_osoba::at(int i){
    return *tab[i];
}
tab_osoba::~tab_osoba(){
    for(int i=0;i<rozmiar;i++)
        delete tab[i];
    delete [] tab;
}
```

Zadanie 1.46

Listing 8.19. rozwiązanie zadania 1.46

```
class kolejka {
private:
    struct element{
        int liczba;
        element * nastepny;
    };
    element *pierw, *ostatni;
public:
    kolejka();
    kolejka(const kolejka&);
    ~kolejka();
    int pierwszy();
    void usun_pierwszy();
    void dodaj_na_koniec(int);
    bool pusta();
};

kolejka::kolejka(){
    pierw=ostatni=NULL;
}

kolejka::kolejka(const kolejka & kol){
    if (kol.pierw==kol.ostatni){
        pierw=ostatni=NULL;
    }
    else {
        element *pom;
        pom=kol.pierw;
        pierw= new element;
        pierw->liczba=pom->liczba;
        ostatni=pierw;
        while(pom!=kol.ostatni){
            pom=pom->nastepny;
            ostatni->nastepny=new element;
            ostatni=ostatni->nastepny;
            ostatni->liczba = pom->liczba;
        }
    }
}

kolejka::~kolejka(){
    if (pierw!=NULL){
        element *pom;
        while(pierw!=ostatni){
            pom=pierw;
            pierw=pierw->nastepny;
            delete pom;
        }
```

```
            delete pierw;
50      }
    }
52
    int kolejka::pierwszy(){
54      return pierw->liczba;
    }
56
    void kolejka::usun_pierwszy(){
58      element* pom=pierw;
        if (pierw==ostatni)
60          pierw=ostatni=NULL;
        else
62          pierw=pierw->nastepny;
        delete pom;
64  }

66
    void kolejka::dodaj_na_koniec(int liczba){
68      if (ostatni!=NULL){
            ostatni->nastepny=new element;
70          ostatni=ostatni->nastepny;
        }
72      else{
            ostatni=pierw=new element;
74      }
        ostatni->liczba=liczba;
76  }

78  bool kolejka::pusta(){
        return (pierw==NULL);
80  }
```

Zadanie 1.57

Listing 8.20. rozwiązanie zadania 1.57

```
    class figura{
2   public:
        double pole, obwod;
4   };

6   class trojkat: public figura{
    public:
8       double a,h;
    };
10
    class prostokat: public figura{
12  public:
        double a,b;
14  };
```

Zadanie 1.58

Listing 8.21. rozwiązanie zadania 1.58

```
struct figura{
public:
    double pole, obwod;
};

struct trojkat: public figura{
public:
    double a,h;
};

struct prostokat: public figura{
public:
    double a,b;
};
```

W przeciwieństwie do klas, które domyślnie dziedziczą w sposób prywatny, struktury dziedziczą domyślnie w sposób publiczny. Klasy mogą dziedziczyć po strukturach, a struktury po klasach. O tym, jaki rodzaj dziedziczenia jest domyślny, decyduje to ,co jest typem pochodnym, struktura czy klasa.

Zadanie 1.62

Listing 8.22. rozwiązanie zadania 1.62

```
class atrakcja {
protected:
    int cena;
    std::string nazwa, opis;
public:
    int WartCena(){return cena;}
    std::string WartNazwa() {return nazwa;}
    std::string WartOpis() {return opis;}
};

class kolejka : public atrakcja{
private:
    std::string godzina_odjazdu, godzina_przyjazdu;
public:
    std::string Odjazd(){return godzina_odjazdu;}
    std::string Przyjazd(){return godzina_przyjazdu;}
    void inicjuj(int, std::string, std::string,
                                    std::string,std::string);
};

void kolejka::inicjuj(int c, std::string n, std::string o,
                          std::string gp, std::string go){
```

```
        cena = c;
24      nazwa = n;
        opis = o;
26      godzina_odjazdu = go;
        godzina_przyjazdu = gp;
28  }
```

Zadanie 1.63

Listing 8.23. rozwiązanie zadania 1.63

```
    class lista {
2   protected:
        struct element{
4           int liczba;
            element * nastepny, *poprzedni;
6       };
        element *pierwszy, *ostatni;
8   public:
        lista();
10      lista(const lista&);
        ~lista();
12      void dodaj_przod(int);
        void dodaj_tyl(int);
14      void usun_przod();
        void usun_tyl();
16      int pierwszy_el();
        int ostatni_el();
18      bool pusta_lista();
    };
20
    lista::lista(){
22      pierwszy=ostatni=NULL;
    }
24
    lista::lista(const lista & list){
26      if (list.pierwszy==list.ostatni){
            pierwszy=ostatni=NULL;
28      }
        else {
30          element *pom;
            pom=list.pierwszy;
32          pierwszy= new element;
            pierwszy->liczba=pom->liczba;
34          ostatni=pierwszy;
            while(pom!=list.ostatni){
36              pom=pom->nastepny;
                ostatni->nastepny=new element;
38              ostatni->nastepny->poprzedni=ostatni;
                ostatni=ostatni->nastepny;
40              ostatni->liczba = pom->liczba;
```

```
            }
42      }
    }
44
    lista::~lista(){
46      if (pierwszy!=NULL){
            element *pom;
48          while(pierwszy!=ostatni){
                pom=pierwszy;
50              pierwszy=pierwszy->nastepny;
                delete pom;
52          }
            delete pierwszy;
54      }
    }
56
    void lista::dodaj_przod(int liczba){
58      if (pierwszy!=NULL){
            pierwszy->poprzedni=new element;
60          pierwszy->poprzedni->nastepny=pierwszy;
            pierwszy=pierwszy->poprzedni;
62      }
        else{
64          ostatni=pierwszy=new element;
        }
66      pierwszy->liczba=liczba;
    }
68
    void lista::dodaj_tyl(int liczba){
70      if (ostatni!=NULL){
            ostatni->nastepny=new element;
72          ostatni->nastepny->poprzedni=ostatni;
            ostatni=ostatni->nastepny;
74      }
        else{
76          ostatni=pierwszy=new element;
        }
78      ostatni->liczba=liczba;
    }
80
    void lista::usun_przod(){
82      element* pom=pierwszy;
        if (pierwszy==ostatni)
84          pierwszy=ostatni=NULL;
        else
86          pierwszy=pierwszy->nastepny;
        delete pom;
88  }

90  void lista::usun_tyl(){
        element* pom=ostatni;
```

```
92      if (pierwszy==ostatni)
            pierwszy=ostatni=NULL;
94      else
            pierwszy=pierwszy->nastepny;
96      delete pom;
    }
98
    int lista::pierwszy_el(){
100     return pierwszy->liczba;
    }
102
    int lista::ostatni_el(){
104     return ostatni->liczba;
    }
106
    bool lista::pusta_lista(){
108     return (pierwszy==NULL);
    }
```

W klasie `lista` stworzyliśmy sekcję chronioną do której dostępu będzie potrzebowało rozwiązanie zadania 1.68.

Zadanie 1.64

Listing 8.24. rozwiązanie zadania 1.64

```
1 class kolejka {
  private:
3     lista list;
  public:
5     int pierwszy() {return list.pierwszy_el();}
      void usun_pierwszy(){list.usun_przod();}
7     void dodaj_na_koniec(int n){list.dodaj_tyl(n);}
      bool pusta(){return list.pusta_lista();}
9 };
```

Zadanie 1.65

Listing 8.25. rozwiązanie zadania 1.65

```
1 class kolejka: private lista {
  public:
3     int pierwszy() {return pierwszy_el();}
      void usun_pierwszy(){usun_przod();}
5     void dodaj_na_koniec(int n){dodaj_tyl(n);}
      bool pusta(){return pusta_lista();}
7 };
```

Jak widać na przykładzie rozwiązań zadań 1.64 i 1.65, dziedziczenie prywatne jest podobne w użyciu do posiadania prywatnego pola danego typu. W zasadzie jedynym istotnym powodem dla którego używa się dziedziczenia prywatnego jest potrzeba dostępu do chronionej sekcji klasy bazowej.

Zadanie 1.68

Listing 8.26. rozwiązanie zadania 1.68

```
class lepsza_lista:public lista{
public:
    class iterator {
    private:
        lepsza_lista& list;
        lista::element *wsk;
     public:
        iterator(lepsza_lista&);
        int& element();
        void nastepny();
        void poprzedni();
        bool poczatek();
        bool koniec();

    };
};

lepsza_lista::iterator::iterator(lepsza_lista & lista):list(lista){
  wsk=list.pierwszy;
}

int& lepsza_lista::iterator::element(){
    return wsk->liczba;
}

void lepsza_lista::iterator::nastepny(){
    wsk=wsk->nastepny;
}

void lepsza_lista::iterator::poprzedni(){
    wsk=wsk->poprzedni;
}

bool lepsza_lista::iterator::poczatek(){
    return wsk==list.pierwszy;
}

bool lepsza_lista::iterator::koniec(){
    return wsk==list.ostatni;
}
```

Zadanie 1.69

Listing 8.27. rozwiązanie zadania 1.69

```
void zeruj(lepsza_lista& list){
    if (!list.pusta_lista()){
        lepsza_lista::iterator it(list);
        while(!it.koniec()){
            it.element()=0;
            it.nastepny();
        }
        it.element()=0;
    }
}
```

Zadanie 1.70

Listing 8.28. rozwiązanie zadania 1.70

```
class stala1{
public:
    const int i;
    stala1():i(5){}
};
```

Miejscem gdzie podajemy wartości stałych pól klasy jest lista inicjalizacyjna konstruktora. Wewnątrz konstruktora jest już za późno na nadawanie wartości stałym polom.

Zadanie 1.71

Listing 8.29. rozwiązanie zadania 1.71

```
class stala2{
public:
    const double d;
    stala2(double dd):d(dd){}
};
```

Zadanie 1.72

Listing 8.30. rozwiązanie zadania 1.72

```
class stale:public stala2{
public:
    stala2 liczba;
    stale(double arg1, double arg2):stala2(arg1),liczba(arg2){}
};
```

Lista inicjalizacyjna to jedyne miejsce, gdzie możemy podać argumenty konstruktora klasy bazowej. Jeżeli klasa bazowa lub któryś z typów pól nie posiada bezargumentowego konstruktora, to wywołanie odpowiedniego konstruktora za argumentami musimy umieścić na liście inicjalizacyjnej naszego konstruktora.

Zadanie 1.80–1.81

Listing 8.31. rozwiązanie zadań 1.80–1.81

```
class liczba{
};

class wymierne: public liczba{
    double d;
};

class calkowite:public liczba{
    int i;
};

liczba* kopiuj(liczba* tab, int n){
    liczba * pom=new liczba[n];
    for(int i=0;i<n;i++)
        pom[i]=tab[i];
    return pom;
}
```

8.2. Rozwiązania zadań z rozdziału 2

Zadanie 2.1

Listing 8.32. rozwiązanie zadania 2.1

```
class bazowa{
public:
    void typ_wskaznika(){
        std::cout<<"bazowa"<<std::endl;
    }

    virtual void typ_obiektu(){
        std::cout<<"bazowa"<<std::endl;
    }
};

class pochodna1: public bazowa{
public:
```

```
    void typ_wskaznika(){
15      std::cout<<"pochodna1"<<std::endl;
    }
17  void typ_obiektu(){
        std::cout<<"pochodna1"<<std::endl;
19  }
};
21
class pochodna2: public bazowa{
23 public:
    void typ_wskaznika(){
25      std::cout<<"pochodna2"<<std::endl;
    }
27  void typ_obiektu(){
        std::cout<<"pochodna2"<<std::endl;
29  }
};
```

Zadanie 2.2

Listing 8.33. rozwiązanie zadania 2.2

```
class liczba{
2 public:
    double re;
4   virtual double modul();
    bool wieksza(liczba&);
6 };

8 double liczba::modul(){
    if (re>=0)
10      return re;
    else
12      return -re;
}
14
bool liczba::wieksza(liczba &ref){
16  if (this->modul()<ref.modul())
        return true;
18  else
        return false;
20 }
```

W zadaniu dla większej przejrzystości kodu użyliśmy wskaźnika `this`. Użycie samej nazwy metody miałoby taki sam skutek.

Zadanie 2.3

Listing 8.34. rozwiązanie zadania 2.3

```
class zespolone: public liczba{
public:
    double im;
    double modul();
};

double zespolone::modul(){
    return sqrt(re*re+im*im);
}
```

Nie ma potrzeby przeciążania metody `wieksza`, gdyż wykorzystuje ona wirtualną metodę `modul`. Pomimo tego, że metoda `wieksza` została zdefiniowana w klasie `liczba`, to używa wersji metody `modul` odpowiedniej dla porównywanych liczb.

Zadanie 2.4

Listing 8.35. rozwiązanie zadania 2.4

```
class funkcja{
public:
    double x;
    virtual double wartosc()=0;
};
```

Zadanie 2.5

Listing 8.36. rozwiązanie zadania 2.5

```
class funkcja_liniowa:public funkcja {
public:
    double a,b;
    double wartosc(){return a*x+b;}
};
```

Zadanie 2.6

Listing 8.37. rozwiązanie zadania 2.6

```
double bisekcja(funkcja * f, double p, double k, double d){
    double wp, wk, ws;
    f->x=p;
    wp=f->wartosc();
    f->x=k;
    wk=f->wartosc();
```

```
7       if (wp*wk<=0)
          while(k-p>d){
9             f->x=(k+p)/2;
              ws=f->wartosc();
11            if (wp*ws<=0)
                  k=f->x;
13            else
                  p=f->x;
15        }
        return p;
17  }
```

Zadanie 2.22

Listing 8.38. rozwiązanie zadania 2.22

```
1  class kolejka{
   public:
3       virtual int pierwszy()=0;
        virtual void usun_pierwszy()=0;
5       virtual void wstaw_na_koniec(int)=0;
        virtual bool pusta()=0;
7       virtual ~kolejka(){}
   };
```

Jeżeli przewidujemy, że po danej klasie mogą dziedziczyć inne klasy, i że będziemy chcieli przy usuwaniu z pamięci obiektów klas pochodnych podać jako argument operatora `delete` wskaźnik do klasy bazowej, to w klasie bazowej powinniśmy zdefiniować wirtualny (ale nie czysto wirtualny) destruktor. Nawet gdyby ten destruktor miał nic nie robić.

Zadanie 2.23

Listing 8.39. rozwiązanie zadania 2.23

```
   class kolejka_listowa: public kolejka {
2  private:
       struct element{
4          int wartosc;
           element * nastepny, * poprzedni;
6      };
       element * pierw, *ost;
8  public:
       kolejka_listowa(){pierw=ost=NULL;}
10     ~kolejka_listowa();
       int pierwszy(){return pierw->wartosc;}
12     void usun_pierwszy();
       void dodaj_na_koniec(int);
14     bool pusta(){return (pierw==NULL);}
```

```
};
kolejka_listowa::~kolejka_listowa(){
    if (pierw!=NULL){
        element * pom=pierw;
        while(pierw!=ost){
            pierw=pierw->nastepny;
            delete pom;
            pom=pierw;
        }
        delete  pom;
    }
}
void kolejka_listowa::usun_pierwszy(){
    if (pierw!=ost){
        element * pom = pierw;
        pierw=pierw->nastepny;
        delete pom;
    }
    else if (pierw!=NULL){
        delete pierw;
        pierw=ost=NULL;
    }
}

void kolejka_listowa::dodaj_na_koniec(int w){
    if (pierw==NULL){
        pierw=ost=new element;
        pierw->wartosc = w;
    }
    else {
        ost->nastepny = new element;
        ost= ost->nastepny;
        ost->wartosc = w;
    }
}
```

8.3. Rozwiązania zadań z rozdziału 3

Zadanie 3.1

Listing 8.40. rozwiązanie zadania 3.1

```
class stale{
    static const double pi=3.1415;
    static const double e=2.7;

};
```

Zadanie 3.2

Listing 8.41. rozwiązanie zadania 3.2

```
class liczba{
public:
    static int licz;
};

int liczba::licz=0;
```

Zadanie 3.6

Listing 8.42. rozwiązanie zadania 3.6

```
class zesp{
public:
    static const zespolone i;

    static zespolone dodaj(zespolone a, zespolone b);
    static zespolone odejmij(zespolone a, zespolone b);
    static zespolone pomnoz(zespolone a, zespolone b);
    static zespolone podziel(zespolone a, zespolone b);
};

const zespolone zesp::i(0,1);

zespolone zesp::dodaj(zespolone a, zespolone b){
    return zespolone(a.re+b.re,a.im+b.im);
}


zespolone zesp::odejmij(zespolone a, zespolone b){
    return zespolone(a.re-b.re,a.im-b.im);
}

zespolone zesp::pomnoz(zespolone a, zespolone b){
    return zespolone(a.re*b.re-a.im*b.im,
                     a.re*b.im+a.im*b.re);
}

zespolone zesp::podziel(zespolone a, zespolone b){
    return zespolone((a.re*b.re+a.im*b.im)/
                      (b.re*b.re+b.im*b.im),
                     (a.im*b.re-a.re*b.im)/
                      (b.re*b.re+b.im*b.im));
}
```

Zadanie 3.7

Listing 8.43. rozwiązanie zadania 3.7

```
#include <iostream>

...

int main(){
    zespolone z(0,1);
    for(int i=0;i<100;i++)
        if (i==0)
            std::cout<<"re = "<<z.re<<" im = "<<z.im
                                                <<std::endl;
        else {
            z= zesp::podziel(zesp::dodaj(
                zesp::pomnoz(zespolone(2,0),z),
                zesp::pomnoz(zespolone(10,0),zesp::i)),z);
            std::cout<<"re = "<<z.re<<" im = "<<z.im
                                                <<std::endl;
        }
}
```

W powyższym rozwiązaniu w miejscu trzech kropek należy wstawić definicję klas `zespolone` i `zesp`.

Zadania 3.8–3.9

Listing 8.44. rozwiązanie zadań 3.8–3.9

```
#include <iostream>

class  zespolone{
public:
    double re,im;
    zespolone(){}
    zespolone(double r, double i):re(r),im(i){}
};

namespace zesp{

const zespolone i(0,1);

zespolone dodaj(zespolone a, zespolone b){
    return zespolone(a.re+b.re,a.im+b.im);
}

zespolone odejmij(zespolone a, zespolone b){
    return zespolone(a.re-b.re,a.im-b.im);
}
```

```
zespolone pomnoz(zespolone a, zespolone b){
    return zespolone(a.re*b.re-a.im*b.im,a.re*b.im+a.im*b.re);
}

zespolone podziel(zespolone a, zespolone b){
    return zespolone((a.re*b.re+a.im*b.im)/(b.re*b.re+b.im*b.im),
                     (a.im*b.re-a.re*b.im)/(b.re*b.re+b.im*b.im));
}
}

int main(){
    zespolone z(0,1);
    for(int i=0;i<100;i++)
        if (i==0)
            std::cout<<"re = "<<z.re<<" im = "<<z.im<<std::endl;
        else {
            z= zesp::podziel(
               zesp::dodaj(zesp::pomnoz(zespolone(2,0),z),
               zesp::pomnoz(zespolone(10,0),zesp::i)),z);
            std::cout<<"re = "<<z.re<<" im = "<<z.im<<std::endl;
        }

}
```

Zadanie 3.16

Listing 8.45. rozwiązanie zadania 3.16

```
class zesp{
private:
    zesp(){}
public:
    static const zespolone i;

    static zespolone dodaj(zespolone a, zespolone b);
    static zespolone odejmij(zespolone a, zespolone b);
    static zespolone pomnoz(zespolone a, zespolone b);
    static zespolone podziel(zespolone a, zespolone b);
};

const zespolone zesp::i(0,1);

zespolone zesp::dodaj(zespolone a, zespolone b){
    return zespolone(a.re+b.re,a.im+b.im);
}


zespolone zesp::odejmij(zespolone a, zespolone b){
    return zespolone(a.re-b.re,a.im-b.im);
}
```

```
24 zespolone zesp::pomnoz(zespolone a, zespolone b){
       return zespolone(a.re*b.re-a.im*b.im,a.re*b.im+a.im*b.re);
26 }

28 zespolone zesp::podziel(zespolone a, zespolone b){
       return zespolone((a.re*b.re+a.im*b.im)/(b.re*b.re+b.im*b.im),
30                      (a.im*b.re-a.re*b.im)/(b.re*b.re+b.im*b.im));
   }
```

Zadanie 3.17

Listing 8.46. rozwiązanie zadania 3.17

```
1 class finalna{
  private:
3     finalna(){}
  public:
5    static finalna * utworz(){ return new finalna;}
  };
```

Po klasie `finalna` nie można dziedziczyć, gdyż z poziomu ewentualnej klasy pochodnej nie byłoby możliwe uruchomienie konstruktora klasy `finalna` (jak pamiętamy, przy tworzeniu obiektu, przed wywołaniem konstruktora klasy pochodnej, wywoływane są konstruktory klas bazowych).

Zauważmy, że ponieważ konstruktor klasy `finalna` jest prywatny, to nie możemy zadeklarować w programie zmiennych tego typu. Obiekty typu `finalna` możemy tworzyć w następujący sposób:

```
finalna *f = finalna::utworz();
```

Standard C++11 oznaczenie słowem `final` klas, po których nie można dziedziczyć oraz metod, których nie można przedefiniowywać w klasach potomnych. W momencie pisania skryptu kompilator `g++` nie posiadał jeszcze zaimplementowanej tej możliwości.

Zadanie 3.18

Listing 8.47. rozwiązanie zadania 3.18

```
1 class dynamiczna{
  protected:
3     dynamiczna(int n){
          tab = new int[n];
5     }
  public:
7     int * tab;
```

```
9       static dynamiczna * utworz(int n)
                {return new dynamiczna(n);}
11  };
```

Zwróćmy uwagę, że dzięki temu, iż klasa `dynamiczna` ma chroniony, a nie prywatny, konstruktor, to można tworzyć jej klasy pochodne.

Zadanie 3.19

Listing 8.48. rozwiązanie zadania 3.19

```
1   class tab_info{
    public:
3       unsigned int w_rozmiar;
        bool czy_rozmiar;
5       int w_wartosc;
        bool czy_wartosc;
7       int w_zmiany;
        bool czy_zmiany;
9       tab_info();
        tab_info& rozmiar(unsigned int);
11      tab_info& wartosc(int);
        tab_info& zmiany(int);
13  };

15  tab_info::tab_info(){
        czy_rozmiar = false;
17      czy_wartosc = false;
        czy_zmiany = false;
19  }

21  tab_info& tab_info::rozmiar(unsigned int n){
        w_rozmiar = n;
23      czy_rozmiar = true;
        return *this;
25  }

27  tab_info& tab_info::wartosc(int n){
        w_wartosc = n;
29      czy_wartosc = true;
        return *this;
31  }

33  tab_info& tab_info::zmiany(int n){
        w_zmiany = n;
35      czy_zmiany = true;
        return *this;
37  }
```

Zadanie 3.20

Listing 8.49. rozwiązanie zadania 3.20

```
class tablica{
private:
    int *tab;
    int *ztab;
    int roz;
public:
    tablica(const tab_info&);
    ~tablica();
    int podaj_w(int);
    void nadaj_w(unsigned int, int);
    int licznik(int);
    int rozmiar();
};

tablica::tablica(const tab_info& t){
    int zm, wart;
    if (t.czy_rozmiar)
        roz=t.w_rozmiar;
    else
        roz=100;
    if (t.czy_wartosc)
        wart=t.w_wartosc;
    else
        wart=0;
    if (t.czy_zmiany)
        zm=t.w_zmiany;
    else
        zm=10;
    tab = new int[roz];
    ztab = new int[roz];

    for(int i=0;i<roz;i++){
        tab[i]=wart;
        ztab[i]=zm;
    }
}

tablica::~tablica(){
    delete [] tab;
    delete [] ztab;
}

int tablica::podaj_w(int i){
    return tab[i];
}

void tablica::nadaj_w(unsigned int i, int w){
    if (ztab[i]!=0)
```

```
            tab[i]=w;
50      if (ztab[i]>0)
            ztab[i]--;
52  }

54  int tablica::licznik(int i){
        return ztab[i];
56  }

58  int tablica::rozmiar(){
        return roz;
60  }
```

Zadanie 3.21

Listing 8.50. rozwiązanie zadania 3.21

```
1  tablica* alokuj(){
       return new tablica(tab_info().rozmiar(50).zmiany(1));
3  }
```

Zadanie 3.24

Listing 8.51. rozwiązanie zadania 3.24

```
1  class napis{
   public:
3      std::string nap;
       bool stala(){return false;}
5      bool stala() const {return true;}
   };
```

Zadanie 3.25

Listing 8.52. rozwiązanie zadania 3.25

```
   class napis2: public std::string{
2  public:
       napis2(){}
4      napis2(const char nap[]):std::string(nap){}
       napis2(const std::string& nap):std::string(nap){}
6      bool stala(){return false;}
       bool stala() const {return true;}
8  };
```

Klasa `napis2` udostępnia nam wszystkie publiczne metody klasy `string` z biblioteki standardowej. Do tego obiekty klasy `napis2` możemy podawać jako argumenty wszędzie tam, gdzie argumentem może być referencja

do typu `string`. Dzięki temu klasę `napis2` możemy traktować jako bogatszą wersję typu `string`.

Zadanie 3.27

Listing 8.53. rozwiązanie zadania 3.27

```
bool porownaj(bazowa* arg1, bazowa *arg2){
    return (typeid(*arg1)==typeid(*arg2));
}
```

Zadanie 3.28

Listing 8.54. rozwiązanie zadania 3.28

```
pochodna * rzutuj(bazowa* b){
    if (typeid(pochodna)==typeid(*b))
        return dynamic_cast<pochodna*>(b);
    else
        return NULL;
}
```

Zadanie 3.30

Listing 8.55. rozwiązanie zadania 3.30

```
void wypisz(liczba2** tab, int n){
    for(int i=0;i<n;i++)
        if (typeid(rzeczywista2)==typeid(tab[i]))
            std::cout<<dynamic_cast<rzeczywista2*>
                        (tab[i])->wartosc<<std::endl;
}
```

8.4. Rozwiązania zadań z rozdziału 4

Zadanie 4.1

Listing 8.56. rozwiązanie zadania 4.1

```
class zespolone{
public:
    double re,im;
};

zespolone operator+(const zespolone& a, const zespolone& b){
    zespolone z;
    z.re=a.re+b.re;
```

```
    z.im=a.im+b.im;
    return z;
}

zespolone operator-(const zespolone& a, const zespolone& b){
    zespolone z;
    z.re=a.re-b.re;
    z.im=a.im-b.im;
    return z;
}

zespolone operator*(const zespolone& a, const zespolone& b){
    zespolone z;
    z.re=a.re*b.re-a.im*b.im;
    z.im=a.im*b.re+a.re*b.im;
    return z;
}

zespolone operator/(const zespolone& a, const zespolone& b){
    zespolone z;
    z.re=(a.re*b.re+a.im*b.im)/(b.re*b.re+b.im*b.im);
    z.im=(a.im*b.re-a.re*b.im)/(b.re*b.re+b.im*b.im);
    return z;
}

std::ostream& operator<<(std::ostream& out,const zespolone& z){
    out<<"re = "<<z.re<<" im = "<<z.im;
    return out;
}

std::istream& operator>>(std::istream& in, zespolone& z){
    in>>z.re>>z.im;
    return in;
}
```

Zadanie 4.4

Listing 8.57. rozwiązanie zadania 4.4

```
class tablica{
private:
    int* tab;
    unsigned int rozmiar;
public:
    tablica();
    tablica(unsigned int);
    tablica(const tablica&);
    ~tablica();
    void resize(unsigned int);
    const tablica& operator=(const tablica&);
    int& operator[](unsigned int);
```

```
    const int& operator[](unsigned int) const;
};

tablica::tablica(){
    tab=NULL;
    rozmiar=0;
}

tablica::tablica(unsigned int n){
    rozmiar=n;
    tab = new int[rozmiar];
}

tablica::tablica(const tablica & t){
    rozmiar = t.rozmiar;
    tab = new int[rozmiar];
    for(unsigned int i=0;i<rozmiar;i++)
        tab[i]=t.tab[i];
}

tablica::~tablica(){
    if (rozmiar>0)
      delete [] tab;
}

void tablica::resize(unsigned int n){
    if (n==0){
        if (rozmiar>0)
          delete [] tab;
        tab=NULL;
    }
    else{
        int * pom = new int[n];
        for(unsigned int i=0;(i<n)&&(i<rozmiar);i++)
            pom[i]=tab[i];
        if (rozmiar>0)
          delete [] tab;
        tab = pom;
    }
    rozmiar = n;
}

const tablica& tablica::operator =(const tablica& t){
    if (rozmiar>0)
        delete [] tab;
    rozmiar=t.rozmiar;
    if (rozmiar==0){
       tab=NULL;
    }
    else {
        tab = new int[rozmiar];
```

```
64          for(unsigned int i=0;i<rozmiar;i++)
                tab[i]=t.tab[i];
66      }
        return t;
68  }

70  int& tablica::operator [](unsigned int i){
        return tab[i];
72  }

74  const int& tablica::operator [](unsigned int i) const{
        return tab[i];
76  }
```

Zadanie 4.6

Listing 8.58. rozwiązanie zadania 4.6

```
    class napis{
2   private:
        char *nap;
4       unsigned int rozm;
    public:
6       napis();
        napis(const char *);
8       napis(const napis&);
        ~napis();
10      const napis& operator=(const napis&);
        bool operator==(const napis&) const;
12      napis operator+(const napis&) const;
        char operator[](unsigned int) const;
14      unsigned int rozmiar() const;
        void wypisz() const;
16  };

18  napis::napis(){
        nap=NULL;
20      rozm=0;
    }
22
    napis::napis(const char * tab){
24      for(rozm=0;tab[rozm]!=0;rozm++);
        if (rozm==0)
26          nap=NULL;
        else{
28        nap=new char[rozm];
          for(unsigned int i=0;i<rozm;i++)
30            nap[i]=tab[i];
        }
32  }
```

```
34 napis::napis(const napis & np){
       rozm=np.rozm;
36     if (rozm==0)
           nap=NULL;
38     else{
           nap=new char[rozm];
40         for(unsigned int i=0;i<rozm;i++)
               nap[i]=np.nap[i];
42     }
   }
44
   napis::~napis(){
46     if (rozm>0)
           delete [] nap;
48 }

50 const napis& napis::operator=(const napis& np){
       if ((rozm>0)&&(rozm!=np.rozm))
52         delete [] nap;
       rozm=np.rozm;
54     if (rozm==0)
           nap=NULL;
56     else{
           nap = new char[rozm];
58         for(unsigned int i=0;i<rozm;i++)
               nap[i]=np.nap[i];
60     }
       return np;
62 }

64 bool napis::operator ==(const napis& np) const{
       if (rozm!=np.rozm)
66         return false;
       for(unsigned int i=0;i<rozm;i++)
68         if (nap[i]!=np.nap[i])
               return false;
70     return true;
   }
72
   napis napis::operator +(const napis& np) const{
74     napis wynik;
       wynik.rozm=rozm+np.rozm;
76     wynik.nap=new char[wynik.rozm];
       unsigned int i=0;
78     for(;i<rozm;i++)
           wynik.nap[i]=nap[i];
80     for(;i<rozm+np.rozm;i++)
           wynik.nap[i]=np.nap[i-rozm];
82     return wynik;
   }
84
```

```
char napis::operator [](unsigned int i) const{
    return nap[i];
}

unsigned int napis::rozmiar() const{
    return rozm;
}

void napis::wypisz() const{
    for(unsigned int i=0;i<rozm;i++)
        std::cout<<nap[i];
    std::cout<<std::endl;
}
```

Zadanie 4.8

Listing 8.59. rozwiązanie zadania 4.8

```
class komperator {
public:
    virtual bool operator()(const napis&, const napis&)=0;
};

class alfabetyczna: public komperator{
public:
    bool operator()(const napis&, const napis&);
};

bool alfabetyczna::operator ()(const napis& np1,
                               const napis& np2){
    unsigned int i=0;
    for(;(i<np1.rozmiar())&&(i<np2.rozmiar());i++)
        if (np1[i]<np2[i])
            return true;
        else if (np1[i]>np2[i])
            return false;
    if (np1.rozmiar()<np2.rozmiar())
        return true;
    return false;
}
```

Zadanie 4.9

Listing 8.60. rozwiązanie zadania 4.9

```
void sortuj(napis* tab, unsigned int n, komperator& komp){
    napis pom;
    for(unsigned int i=0;i<n-1;i++)
        for(unsigned int j=0;j<n-i-1;j++)
            if (komp(tab[j+1],tab[j])){
```

```
6                   pom=tab[j];
                    tab[j]=tab[j+1];
8                   tab[j+1]=pom;
                }
10 }
```

Zadanie 4.12

Listing 8.61. rozwiązanie zadania 4.12

```
class macierz{
2 private:
    double ** tab;
4 public:
    macierz(unsigned int, unsigned int);
6   double& operator()(unsigned int, unsigned int);
};
8
macierz::macierz(unsigned int n, unsigned int m){
10  tab = new double*[n];
    for(unsigned int i=0;i<m;i++)
12      tab[i] = new double[m];
}
14
double& macierz::operator ()(unsigned int i, unsigned int j){
16  return tab[i][j];
}
```

W przeciwieństwie do operatora `[]` operator `()` nie ma z góry kreślonej arności i dzięki temu mogliśmy go przeciążyć tak, żeby na raz podawać mu oba indeksy w dwuwymiarowej tablicy. Osiągnięcie podobnego efektu w przypadku operatora `[]` jest możliwe, ale jest to bardziej skomplikowane.

Zadanie 4.14

Listing 8.62. rozwiązanie zadania 4.14

```
1 struct punkt2D{
    double x,y;
3 };

5 struct punkt3D{
    double x,y,z;
7
    operator punkt2D() const;
9 };

11 punkt3D::operator punkt2D() const{
    punkt2D punkt;
```

```
13      punkt.x=x;
        punkt.y=y;
15      return punkt;
    }
```

Zadanie 4.18

Listing 8.63. rozwiązanie zadania 4.18

```
    struct elisty{
2   struct element{
      element * nastepny, poprzedni;
4     int i;
    };
6
    element * wsk, *pierwszy, *ostatni;
8
    elisty& operator++(); //wersja prefiksowa
10  elisty operator++(int); //wersja postfiksowa
    elisty& operator--(); //wersja prefiksowa
12  elisty operator--(int); //wersja postfiksowa
    };
14
    elisty& elisty::operator ++(){
16      if (wsk!=ostatni)
           wsk=wsk->nastepny;
18      return *this;
    }
20
    elisty elisty::operator ++(int i){
22      elisty pom=*this;
        if (wsk!=ostatni)
24         wsk=wsk->nastepny;
        return pom;
26  }

28  elisty& elisty::operator--(){
        if (wsk!=pierwszy)
30         wsk=wsk->poprzedni;
        return *this;
32  }

34  elisty elisty::operator --(int i){
        elisty pom=*this;
36      if (wsk!=pierwszy)
           wsk=wsk->poprzedni;
38      return pom;
    }
```

Zadanie 4.21

Listing 8.64. rozwiązanie zadania 4.21

```
class n_int {
private:
    static unsigned int liczba_ob, liczba_tab;

public:
    int liczba;
    void* operator new(size_t);
    void* operator new[](size_t);
    void operator delete(void*);
    void operator delete[](void*);
    static void wypisz();
};

unsigned int n_int::liczba_ob=0, n_int::liczba_tab=0;


void* n_int::operator new(size_t rozmiar) {
   liczba_ob++;
   return malloc(rozmiar);
}

void* n_int::operator new[](size_t rozmiar) {
    liczba_tab++;
    return malloc(rozmiar);
}

void n_int::operator delete(void* wsk){
    liczba_ob--;
    free(wsk);
}

void n_int::operator delete[](void* wsk){
    liczba_tab--;
    free(wsk);
}

void n_int::wypisz(){
    std::cout<<"Zaalokowano dynamicznie "<<liczba_ob
             <<" pojedynczych obiektow typu n_int"
             <<std::endl;
    std::cout<<"oraz "<<liczba_tab<<" dynamicznych tablic tego typu.";
}
```

Dla uproszczenia w powyższym rozwiązaniu nie uwzględniliśmy faktu, że standardowy operator `new` nie zwraca nigdy wartości `NULL`, a w przypadku niepowodzenia alokacji pamięci rzuca wyjątek. Warto pamiętać,

że standardowy operator `new` posiada specjalną wersję nierzucająca wyjątków.

8.5. Rozwiązania zadań z rozdziału 5

Zadanie 5.1

Listing 8.65. rozwiązanie zadania 5.1

```
template<class T>
T funkcja(T a, T b, T c){
    return a-b+c;
}
```

Zadanie 5.4

Listing 8.66. rozwiązanie zadania 5.4

```
template<class T>
T najmniejszy(T* tab, unsigned int n){
    unsigned int min=0;
    for(unsigned int i=1;i<n;i++)
        if (tab[i]<tab[min])
            min=i;
    return tab[min];
}
```

Zadanie 5.5

Listing 8.67. rozwiązanie zadania 5.5

```
template<class T>
void zamien(T& a, T& b){
    T c=a;
    a=b;
    b=c;
}
```

Zadanie 5.6

Listing 8.68. rozwiązanie zadania 5.6

```
template<class T>
void wypisz(T* a){
    std::cout<<*a<<std::endl;
}
```

Zadanie 5.7

Listing 8.69. rozwiązanie zadania 5.7

```
template<class T>
void wypisz_zakres(T* pocz, T* kon){
    while(pocz!=kon){
        wypisz<T>(pocz);
        pocz++;
    }
}
```

W rozwiązaniu wywołując szablon `wypisz`, podaliśmy wprost, choć nie musieliśmy, parametr szablonu. Często, choć nie zawsze, możemy pominąć parametry przy wywoływaniu szablonu funkcji (kompilator dedukuje wówczas parametry szablonu z typów argumentów).

Zadanie 5.8

Listing 8.70. rozwiązanie zadania 5.8

```
template<class T>
void wypisz_tab(T * tab, unsigned int n){
    wypisz_zakres<T>(tab, tab+n);
}
```

Zadanie 5.9

Listing 8.71. rozwiązanie zadania 5.9

```
template<class T>
void wypisz(T a){
    std::cout<<*a<<std::endl;
}

template<class T>
void wypisz_zakres(T pocz, T kon){
    while(pocz!=kon){
        wypisz(pocz);
        pocz++;
    }
}

template<class T>
void wypisz_tab(T tab, unsigned int n){
    wypisz_zakres(tab, tab+n);
}

template<class T>
void wypisz_vec(const std::vector<T>& vec){
```

```
    wypisz_zakres(vec.begin(),vec.end());
22 }
```

Zadanie 5.10

Listing 8.72. rozwiązanie zadania 5.10

```
1 template<class T>
  bool operator>(const T& a, const T& b){
3     return !(a<b);
  }
```

Zadanie 5.11

Listing 8.73. rozwiązanie zadania 5.11

```
  template<class T1, class T2>
2 bool typ(T1* a, T2* b){
      return (typeid(*a)==typeid(*b));
4 }
```

Zadanie 5.12

Listing 8.74. rozwiązanie zadania 5.12

```
  template<class T1, class T2>
2 void minimum(T1 tab[],unsigned int n, T2 f){
      unsigned int min;
4     for(int i=1;i<n;i++)
          if (f(tab[i],tab[min]))
6             min=i;
      return tab[min];
8 }
```

Zadanie 5.13

Listing 8.75. rozwiązanie zadania 5.13

```
  class mniejszy {
2 public:
      bool operator()(int a, int b){return (a<b);}
4 };

6 class wiekszy {
  public:
8     bool operator()(int a, int b){return (a>b);}
  };
10
  void skrajne(int tab[],unsigned int n){
```

```
12      std::cout<<"Najmniejszy element tablicy to "
                 <<minimum(tab,n,mniejszy())<<std::endl;
14      std::cout<<"Najwiekszy element tablicy to "
                 <<minimum(tab,n,wiekszy())<<std::endl;
16  }
```

Zadanie 5.16

Listing 8.76. rozwiązanie zadania 5.16

```
  template<unsigned int n>
2 void przepisz(int m1[][n], int m2[][n]){
      for(unsigned int i=0;i<n;i++)
4         for(unsigned int j=0;j<n;j++)
              m2[i][j]=m1[i][j];
6 }
```

Zauważmy, że w języku C++ nie możemy tworzyć wielowymiarowych tablic automatycznych o rozmiarach podanych przez zmienne. Szablony takie jak powyżej pozwalają nam obejść część związanych z tym niedogodności, ale nie pozwalają na tworzenie tablic o rozmiarze nieznanym w trakcie kompilacji programu.

Zadanie 5.17

Listing 8.77. rozwiązanie zadania 5.17

```
  class tablica{
2 public:
      int * tab;
4     unsigned int rozmiar_tab;

6     tablica(unsigned int);
      ~tablica();
8     template<class T>
      void sortuj(T);
10 };

12 tablica::tablica(unsigned int n){
      tab = new int[n];
14    rozmiar_tab = n;
  }
16
  tablica::~tablica(){
18    delete [] tab;
  }
20
  template<class T>
22 void tablica::sortuj(T f){
```

```
    int pom;
24  for(unsigned int i=0;i<rozmiar_tab-1;i++)
        for(unsigned int j=0;j<rozmiar_tab-i-1;j++)
26          if (f(tab[j+1],tab[j])){
                pom=tab[j];
28              tab[j]=tab[j+1];
                tab[j+1]=pom;
30          }
}
```

Zadanie 5.22

Listing 8.78. rozwiązanie zadania 5.22

```
1  template<class T>
   class liczba {
3  private:
       T wart;
5  public:
       void wczytaj();
7      void wypisz();
   };
9
   template<class T>
11 void liczba<T>::wczytaj(){
       std::cout<<"Podaj wartosc ";
13     std::cin>>wart;
   }
15
   template <class T>
17 void liczba<T>::wypisz(){
       std::cout<<" Przechowywana wartosc to "<<wart<<std::endl;
19 }
```

Zadanie 5.25

Listing 8.79. rozwiązanie zadania 5.25

```
1  template<class T1, class T2>
   class para{
3  public:
       T1 pierwsze;
5      T2 drugie;

7      para(T1, T2);
   };
9
   template<class T1, class T2>
11 para<T1,T2>::para(T1 a, T2 b){
       pierwsze = a;
```

```
13      drugie = b;
    }
15
    template<class T1, class T2>
17  bool operator<(para<T1,T2> a, para<T1,T2> b){
        if (a.pierwsze<b.pierwsze)
19          return true;
        else if ((a.pierwsze==b.pierwsze)&&(a.drugie<b.drugie))
21          return true;
        return false;
23  }
```

Zadanie 5.27

Listing 8.80. rozwiązanie zadania 5.27

```
1  template<class T1,unsigned int n>
   class tablica2 {
3  public:
       T1 tab[n];
5      static const unsigned int rozmiar = n;
   };
```

Ponieważ tablica `tab` jest tablicą automatyczną, jest ona poprawnie kopiowana przez standardowy konstruktor kopiujący i operator przypisania, zaś pamięć po niej jest automatycznie zwalniana w trakcie zwalniania pamięci przez obiekt, który ją przechowuje. W związku z tym nie ma potrzeby definiowania w szablonie klasy `tablica2` konstruktora kopiującego, operatora przypisania czy destruktora.

Zadanie 5.28

Listing 8.81. rozwiązanie zadania 5.28

```
    template<unsigned int n>
2   class wektor {
    public:
4       double tab[n];
    };
6
    template<unsigned int n>
8   wektor<n> operator+(const wektor<n>& a, const wektor<n>& b){
        wektor<n> wynik;
10      for(unsigned int i=0;i<n;i++)
            wynik.tab[i]=a.tab[i]+b.tab[i];
12      return wynik;
    }
14
    template<unsigned int n>
```

```
16 wektor<n> operator-(const wektor<n>& a, const wektor<n>& b){
       wektor<n> wynik;
18     for(unsigned int i=0;i<n;i++)
           wynik.tab[i]=a.tab[i]-b.tab[i];
20     return wynik;
   }
22
   template<unsigned int n>
24 wektor<n> operator*(int a, const wektor<n>& b){
       wektor<n> wynik;
26     for(unsigned int i=0;i<n;i++)
           wynik.tab[i]=a*b.tab[i];
28     return wynik;
   }
```

Zwróćmy uwagę, że w powyższym przykładzie nie potrzebujemy przechowywać ilości wymiarów wektora w polu klasy, gdyż jest to parametr szablonu. Co więcej, próba dodania do siebie wektorów o różnej liczbie wymiarów zostanie wychwycona już na etapie kompilacji.

Zadania 5.29–5.31

Listing 8.82. rozwiązanie zadań 5.29–5.31

```
1  template<unsigned int n>
   class punkt {
3  public:
       double tab[n];
5
       punkt(){}
7      punkt(const punkt<n+1>&);
   };
9
   template<unsigned int n>
11 punkt<n>::punkt(const punkt<n+1> & p){
       for(unsigned int i=0;i<n;i++)
13         tab[i]=p.tab[i];
   }
15
   template<unsigned int n>
17 punkt<n-1> zrzutuj(const punkt<n>& p){
       punkt<n-1> wynik;
19     for(unsigned int i=0;i<n-1;i++)
           wynik.tab[i]=p.tab[i];
21     return wynik;
   }
```

Funkcja `zrzutuj` jest wywoływana tylko wtedy gdy ją jawnie wywołamy. W przypadku zdefiniowanego powyżej konstruktora może być on wywołany

także niejawnie (na przykład gdy po lewej stronie operatora przypisania jest zmienna typu `punkt<n>` a po prawej typu `punkt<n+1>`). Niejawne wywoływanie konstruktora może być wygodne, ale niesie za sobą także pewne niebezpieczeństwa (przykładowo możemy nie zauważyć, że dokonaliśmy przypisania pomiędzy punktami o różnej liczbie współrzędnych w wyniku czego utraciliśmy wartość ostatniej współrzędnej).

Zadanie 5.38

Listing 8.83. rozwiązanie zadania 5.38

```
template<class T>
class lista {
public:
    virtual void wstaw_z_przodu(T)=0;
    virtual T pierwszy()=0;
    virtual void usun_pierwszy()=0;
    virtual void wstaw_z_tylu(T)=0;
    virtual T ostatni()=0;
    virtual void usun_ostatni()=0;
    virtual bool pusta()=0;
    virtual ~lista(){}
};
```

Zadanie 5.39

Listing 8.84. rozwiązanie zadania 5.39

```
template<class T>
class lista_wskaz: public lista<T>{
private:
    struct element {
        T wartosc;
        element * nastepny, *poprzedni;
    };
    element *pierw, *ost;
public:
    lista_wskaz();
    ~lista_wskaz();
    void wstaw_z_przodu(T);
    T pierwszy();
    void usun_pierwszy();
    void wstaw_z_tylu(T);
    T ostatni();
    void usun_ostatni();
    bool pusta();
};
template<class T>
lista_wskaz<T>::lista_wskaz(){
    pierw=ost=NULL;
```

```
}

template<class T>
lista_wskaz<T>::~lista_wskaz(){
    if (pierw!=NULL){
        element* pom;
        while(pierw!=ost){
            pom=pierw;
            pierw=pierw->nastepny;
            delete pom;
        }
        delete pierw;
    }
}

template<class T>
void lista_wskaz<T>::wstaw_z_przodu(T t){
    if (pierw==NULL){
        pierw=ost=new element;
    }
    else{
      pierw->poprzedni = new element;
      pierw=pierw->poprzedni;
    }
    pierw->wartosc = t;
}

template<class T>
T lista_wskaz<T>::pierwszy(){
    return pierw->wartosc;
}


template<class T>
void lista_wskaz<T>::usun_pierwszy(){
    element *pom=pierw;
    if (pierw==ost)
        pierw=ost=NULL;
    else
        pierw=pierw->nastepny;
    delete pom;
}

template<class T>
void lista_wskaz<T>::wstaw_z_tylu(T t){
    if (ost==NULL){
        pierw=ost=new element;
    }
    else{
      ost->nastepny = new element;
      ost=ost->nastepny;
```

```
74      }
        ost->wartosc = t;
76  }

78  template<class T>
    T lista_wskaz<T>::ostatni(){
80      return ost->wartosc;
    }
82

84  template<class T>
    void lista_wskaz<T>::usun_ostatni(){
86      element *pom=ost;
        if (pierw==ost)
88          pierw=ost=NULL;
        else
90          ost=ost->poprzedni;
        delete pom;
92  }

94  template<class T>
    bool lista_wskaz<T>::pusta(){
96      return (pierw==NULL);
    }
```

Zadanie 5.42

Listing 8.85. rozwiązanie zadania 5.42

```
1   template<class T>
    class kolejka {
3   private:
        lista<T> * list;
5   public:
        kolejka(lista<T>* l){list = l;}
7       kolejka(){list = new lista_wskaz<T>;}
        ~kolejka(){delete list;}
9       T pierwszy(){return list->pierwszy();}
        void usun_pierwszy(){list->usun_pierwszy();}
11      void dodaj_na_koniec(T t){list->wstaw_z_tylu(t);}
        bool pusta(){return list->pusta();}
13  };
```

Zadanie 5.43

Listing 8.86. rozwiązanie zadania 5.43

```
1   template<class T>
    class tablica {
3   private:
```

```
    T * tab;
    unsigned int roz;
public:
    tablica();
    tablica(unsigned int);
    tablica(const tablica&);
    ~tablica();
    unsigned int rozmiar() const;
    const tablica & operator=(const tablica&);
    T& operator[](unsigned int);
    const T& operator[](unsigned int) const;
};

template<class T>
tablica<T>::tablica(){
    roz=0;
}

template<class T>
tablica<T>::tablica(unsigned int n){
    roz=n;
    tab = new T[roz];
}

template<class T>
tablica<T>::tablica(const tablica<T> & t){
    roz=t.roz;
    tab = new T[roz];
    for(unsigned int i=0;i<roz;i++)
        tab[i]=t.tab[i];
}


template<class T>
tablica<T>::~tablica(){
    delete [] tab;
}

template<class T>
unsigned int tablica<T>::rozmiar() const{
    return roz;
}

template<class T>
const tablica<T>& tablica<T>::operator=(const tablica<T>& t){
    if (roz>0)
        delete[] tab;
    roz=t.roz;
    tab = new T[roz];
    for(unsigned int i=0;i<roz;i++)
        tab[i]=t.tab[i];
```

```
55      return t;
    }
57
    template<class T>
59  T& tablica<T>::operator[](unsigned int i){
        return tab[i];
61  }

63  template<class T>
    const T& tablica<T>::operator[](unsigned int i) const{
65      return tab[i];
    }
```

Zadanie 5.44

Listing 8.87. rozwiązanie zadania 5.44

```
    typedef tablica<char> napis;
2
    napis operator+(const napis& a,const napis& b ){
4       napis c(a.rozmiar()+b.rozmiar());
        for(unsigned int i=0;i<a.rozmiar();i++)
6           c[i]=a[i];
        for(unsigned int i=a.rozmiar();i<c.rozmiar();i++)
8           c[i]=b[i-a.rozmiar()];
        return c;
10  }
```

Zadanie 5.46

Listing 8.88. rozwiązanie zadania 5.46

```
    template<class T>
2   class tablica {
    private:
4       T * tab;
        unsigned int roz;
6   public:
        tablica();
8       tablica(unsigned int);
        tablica(const tablica&);
10      ~tablica();
        unsigned int rozmiar() const;
12      const tablica & operator=(const tablica&);
        T& operator[](unsigned int);
14      const T& operator[](unsigned int) const;
        template<class R>
16      bool operator==(const tablica<R>&) const;
    };
18
```

```
template<class T>
template<class R>
bool tablica<T>::operator==(const tablica<R> & t) const{
    if (roz!=t.rozmiar())
        return false;
    for(unsigned int i=0; i<roz;i++)
        if (tab[i]!=t[i])
            return false;
    return true;
}
```

Pominęliśmy definicje wszystkich klas poza operatorem `==`, gdyż czytelnik może je znaleźć w rozwiązaniu zadania 5.43. Zwróćmy uwagę na dwie rzeczy. Po pierwsze w definicji operatora mamy szablon szablonów, a nie jeden szablon o dwóch parametrach. Po drugie w ciele operatora używamy wyłącznie publicznych metod obiektu t. Dzieje się tak, gdyż dla `T` różnego od `R`, `tablica<T>` i `tablica<R>` to dwa różne typy, a więc metody klasy `tablica<T>` nie mają dostępu do prywatnych pól klasy `tablica<R>`. Zauważmy też, że powyższy operator porównania działa także dla klasy `napis` z zadania 5.44.

Zadanie 5.48

Listing 8.89. zawartość pliku glowny.cc

```
#include <iostream>
#include "tablica.h"

int main(){
    unsigned int n,pom;
    std::cout<<"ile liczb chcesz podac ";
    std::cin>>n;
    tablica<int> tab(n);
    std::cout<<"podaj "<<n<<" liczb :"<<std::endl;
    for(unsigned int i=0;i<n;i++)
        std::cin>>tab[i];

    for(unsigned int i=0;i<tab.rozmiar()-1;i++)
        for(unsigned int j=0;j<tab.rozmiar()-i-1;j++)
            if (tab[j]>tab[j+1]){
                pom=tab[j];
                tab[j]=tab[j+1];
                tab[j+1]=pom;
            }
    for(unsigned int i=0;i<tab.rozmiar();i++)
        std::cout<<tab[i]<<std::endl;

}
```

Całe rozwiązanie zadania 5.43 powinno zostać umieszczone w pliku `tablica.h`. Modułów z szablonami nie rozdziela się na plik nagłówkowy i kod, gdyż szablony muszą być kompilowane razem z jednostkami kompilacji, w których są użyte (inaczej kompilator nie wie, jakie konkretyzacje będą potrzebne).

8.6. Rozwiązania zadań z rozdziału 6

Zadanie 6.1

Listing 8.90. rozwiązanie zadania 6.1

```
#include <iostream>
#include <vector>

int main(){
    std::vector<int> v;
    int liczba;
    do {
      std::cin>>liczba;
      v.push_back(liczba);
    } while(liczba);

    for(unsigned int i=0;i<v.size();i++)
        std::cout<<v[i]<<std::endl;
}
```

Zadanie 6.3 Poniżej rozwiązani poszczególnych podpunktów zadania 6.3:

Listing 8.91. rozwiązanie zadania 6.3a

```
void pomieszaj(std::vector<int>& v){
    std::reverse(v.begin(),v.end());
}
```

Listing 8.92. rozwiązanie zadania 6.3b

```
void pomieszaj(std::vector<int>& v){
    std::sort(v.begin(),v.end());
}
```

Listing 8.93. rozwiązanie zadania 6.3c

```
void pomieszaj(std::vector<int>& v){
    std::sort(v.begin(),v.end(),std::greater<int>());
}
```

Listing 8.94. rozwiązanie zadania 6.3d

```
class wartosc_bezwzgledna{
public:
    bool operator()(int a, int b){
        if (a<0)
            a=-a;
        if (b<0)
            b=-b;
        return (a<b);
    }
};

void pomieszaj(std::vector<int>& v){
    std::sort(v.begin(),v.end(),wartosc_bezwzgledna());
}
```

Listing 8.95. rozwiązanie zadania 6.3e

```
class reszta{
public:
    bool operator()(int a, int b){

        return (a%1000<b%1000);
    }
};

void pomieszaj(std::vector<int>& v){
    std::sort(v.begin(),v.end(),reszta());
}
```

Zadanie 6.4 W przypadku gdy funkcja ma działać dla kontenera `array` o okreslonym z góry rozmiarze, jego użycie nie różni się od użycia wektora:

Listing 8.96. rozwiązanie zadania 6.4

```
void pomieszaj(std::array<int,10>& v){
    std::reverse(v.begin(),v.end());
}
```

Jednak, gdy chcemy aby funkcję można było użyć dla kontenerów o różnych rozmiarach musimy stworzyć szablon funkcji

Listing 8.97. rozwiązanie zadania 6.4

```
template<unsigned int n>
void pomieszaj(std::array<int,n>& v){
    std::reverse(v.begin(),v.end());
}
```

Zadanie 6.5

Listing 8.98. rozwiązanie zadania 6.5

```
class macierz {
private:
    std::vector<std::vector<int> > v;
public:
    macierz(unsigned int n, unsigned int m):
                    v(n,std::vector<int>(m)){}
    int& operator()(unsigned int i, unsigned  int j)
                    {return v[i][j];}
};
```

Zadanie 6.9

Listing 8.99. rozwiązanie zadania 6.9

```
#include <iostream>
#include <list>

int main(){
    std::list<int> l;
    unsigned int n;
    int pom;
    std::cin>>n;
    for(unsigned int i=0;i<n;i++){
        std::cin>>pom;
        if (pom>=0)
            l.push_back(pom);
        else
            l.push_front(pom);
    }
    for(std::list<int>::iterator it=l.begin();it!=l.end();it++)
        std::cout<<*it<<std::endl;
}
```

Zadanie 6.12

Listing 8.100. rozwiązanie zadania 6.12

```
class lista_napisow {
private:
    std::queue<std::string, std::list<std::string> > kolejka;
public:
    void wczytaj();
    void wypisz();
};

void lista_napisow::wczytaj(){
    std::string s;
```

```
    std::cin>>s;
12      kolejka.push(s);
}
14
void lista_napisow::wypisz(){
16      for(unsigned int i=0;(i<10)&&(!kolejka.empty());i++){
            std::cout<<kolejka.front()<<std::endl;
18          kolejka.pop();
        }
20  }
```

Zadanie 6.19

Listing 8.101. rozwiązanie zadania 6.19

```
class porownaj {
2   public:
        porownaj(){}
4       bool operator()(const dane& d1, const dane& d2){
            if (d1.punkty<d2.punkty)
6               return true;
            else if ((d1.punkty==d2.punkty)&&(d1.pesel<d2.pesel))
8               return true;
            else
10              return false;

12      }
};
14
class rekrutacja {
16   private:
         std::priority_queue<dane,
18           std::vector<dane>, porownaj> kolejka;
     public:
20       void dodaj(std::string, std::string,std::string,
                                  std::string, unsigned int);
22       dane najlepszy();
         void usun();
24   };

26  void rekrutacja::dodaj(std::string imie,
          std::string nazwisko, std::string pesel,
28        std::string tel, unsigned int punkty){
        dane pom;
30      pom.imie = imie;
        pom.nazwisko = nazwisko;
32      pom.pesel = pesel;
        pom.tel = tel;
34      pom.punkty = punkty;
        kolejka.push(pom);
36  }
```

```
38 dane rekrutacja::najlepszy(){
       return kolejka.top();
40 }

42 void rekrutacja::usun(){
       kolejka.pop();
44 }
```

Przy rozwiązywaniu zadania założyliśmy, że nie może być w bazie dwóch różnych osób o takim samym numerze pesel.

Zadania 6.24–6.25

Listing 8.102. rozwiązanie zadań 6.24–6.25

```
1  class slownik {
   private:
3      std::set<std::string> zbior;
   public:
5      void dodaj(const std::string&);
       bool znajdz(const std::string&);
7      void zakres(const std::string&, const std::string&);
   };
9
   void slownik::dodaj(const std::string& slowo){
11     zbior.insert(slowo);
   }
13
   bool slownik::znajdz(const std::string& slowo){
15     return (zbior.find(slowo)!=zbior.end());
   }
17
   void slownik::zakres(const std::string& slowo1,
19                       const std::string& slowo2){
       std::set<std::string>::iterator pocz, kon;
21     pocz=zbior.upper_bound(slowo1);
       kon=zbior.lower_bound(slowo2);
23     while(pocz!=kon){
           std::cout<<*pocz<<std::endl;
25         pocz++;
       }
27 }
```

Zadanie 6.32

Listing 8.103. rozwiązanie zadania 6.32

```
1  template<class T>
   void wypisz(T pocz, T kon){
```

```
3       while(pocz!=kon){
            std::cout<<*pocz<<std::endl;
5           pocz++;
        }
7   }
```

Zadanie 6.38

Listing 8.104. rozwiązanie zadania 6.38

```
1   struct bezwzgledna{
        void operator()(int& i){
3           if (i<0)
                i=-i;
5       }
    };
7
    void vec_abs(std::vector<int>& v){
9       std::for_each(v.begin(),v.end(),bezwzgledna());
    }
```

Zadanie 6.39 W rozwiązaniach wszystkich podpunktów będziemy wykorzystywali następującą strukturę:

Listing 8.105. rozwiązanie zadania 6.39

```
    struct podzielny{
2     bool operator()(int i) const{
        if ((i%2==0)||(i%3==0)||(i%5==0)||(i%7==0))
4         return true;
        return false;
6     }
    };
```

Poniżej rozwiązania dla poszczególnych podpunktów:

Listing 8.106. rozwiązanie zadania 6.39a

```
1   bool ktorykolwiek(const std::vector<int>& v){
      return any_of(v.begin(),v.end(),podzielny());
3   }
```

Listing 8.107. rozwiązanie zadania 6.39b

```
1   bool zaden(const std::vector<int>& v){
      return none_of(v.begin(),v.end(),podzielny());
3   }
```

Listing 8.108. rozwiązanie zadania 6.39c

```
bool kazdy(const std::vector<int>& v){
  return all_of(v.begin(),v.end(),podzielny());
}
```

Zadanie 6.42

Listing 8.109. rozwiązanie zadania 6.42

```
bool palindrom(std::string s){
    return std::equal(s.begin(),s.end(),s.rbegin());
}
```

W rozwiązaniu tego zadania warto zwrócić uwagę na użycie `v.rbegin()` – iteratora poruszającego się od końca (`reverse_iterator`).

8.7. Rozwiązania zadań z rozdziału 7

Zadanie 7.1

Listing 8.110. rozwiązanie zadania 7.1

```
double* alokuj_wer_1(unsigned int n){
    try {
        return new double[n];
    }
    catch(...){
        std::cout<<"Nieudana proba alokacji tablicy"
                                            <<std::endl;
        return NULL;
    }
}

double* alokuj_wer_2(unsigned int n){
    double * pom = new (std::nothrow) double[n];
    if (pom==NULL)
        std::cout<<"Nieudana proba alokacji tablicy"
                                            <<std::endl;
    return pom;
}
```

Zadanie 7.2

Listing 8.111. rozwiązanie zadania 7.2

```
void zamien(std::vector<int> v1,std::vector<int> v2,
            std::vector<int> v3, unsigned int l1,
                unsigned int l2, unsigned int l3){
```

```
4       if ((v1.size()<=l1)||(v2.size()<=l2)||(v3.size()<=l3))
            throw std::out_of_range("indeks spoza zakresu");
6       else {
            int pom;
8           pom = v1[l1];
            v1[l1]=v3[l3];
10          v3[l3]=v2[l2];
            v2[l2]=pom;
12      }
    }
```

Zadanie 7.4

Listing 8.112. rozwiązanie zadania 7.4

```
1  class blad {
       std::string opis;
3  public:
       blad(const std::string& s):opis(s){}
5      std::string co(){return opis;}
   };
7
   class spoza_dziedziny: public blad{
9  public:
       spoza_dziedziny(std::string s):blad(s){}
11     spoza_dziedziny():blad("argumenty poza dziedziny"){}
   };
13
   class dzielenie_przez_zero: public spoza_dziedziny{
15 public:
       dzielenie_przez_zero():
17     spoza_dziedziny("dzielenie przez zero"){}
   };
19
   class pierwiastek_z_ujemnej: public spoza_dziedziny {
21 public:
       pierwiastek_z_ujemnej():
23     spoza_dziedziny("pierwiastek z liczby ujemnej"){}
   };
25

27 class niezainicjowana_zmienne: public blad {
   public:
29     niezainicjowana_zmienne():
       blad("uzycie niezainicjowanej zmiennej"){}
31 };

33
   class przypisanie_do_stalej: public blad {
35 public:
       przypisanie_do_stalej():
```

```
37      blad("proba przypisania nowej wartosci do stalej"){}
};
39
class wynik_spoza_zakresu: public blad {
41 public:
    wynik_spoza_zakresu():
43      blad("wynik spoza zakresu uzytego typu"){}
};
45
class inny_blad: public blad {
47 public:
    inny_blad():blad("inny blad"){}
49 };
```

Powyżej stworzyliśmy swoją pełną hierarchię wyjątków. W praktyce dobrym pomysłem jest umieszczenie swoich wyjątków w hierarchii standardowych wyjątków poprzez dziedziczenie po odpowiednich klasach wyjątków.

Zadanie 7.5

Listing 8.113. rozwiązanie zadania 7.5a

```
1 double pot(double p, double w){
    if ((!std::isfinite(p))||(!std::isfinite(w)))
3       throw spoza_dziedziny();
    if ((p==0)&&(w<=0))
5       throw spoza_dziedziny();
    if ((p<0)&&(w!=trunc(w)))
7       throw spoza_dziedziny();
    errno = 0;
9   double pom = pow(p,w);
    if (errno==ERANGE)
11      throw wynik_spoza_zakresu();
    if (errno==EDOM)
13          throw inny_blad();
    return pom;
15 }
```

Listing 8.114. rozwiązanie zadania 7.5b

```
1 double pierw(double p){
    if (!std::isfinite(p))
3       throw spoza_dziedziny();
    errno = 0;
5   if (p<0)
        throw pierwiastek_z_ujemnej();
7   double pom= sqrt(p);
    if (errno==ERANGE)
9       throw wynik_spoza_zakresu();
```

```
    if (errno==EDOM)
11      throw spoza_dziedziny();
    return pom;
13 }
```

W powyższych przypadkach nie mogliśmy w pełni polegać na kodach zapisanych w `errno`, gdyż nasze klasyfikacja błędów nie pokrywa się z tą stosowaną przez funkcje standardowe (nie tylko dlatego, że nasza klasyfikacja jest bardziej szczegółowa).

W rozwiązaniu użyliśmy funkcji `isfinite` sprawdzającej czy wartością podaną w jej argumencie nie jest `inf` ani `nan`. Funkcja `isfinite` została dodana oficjalnie dopiero w standardzie C++11x, ale już wcześniej była obsługiwana przez wiele kompilatorów (w języku C ta funkcja jest dostępna od standardu C99).

Zadanie 7.6

Listing 8.115. rozwiązanie zadania 7.6a

```
1 double dodaj(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b))){
3       errno = EDOM;
        return 0;
5   }
    double c= a+b;
7   if (!std::isfinite(c))
        errno = ERANGE;
9   return c;
  }
11
  double odejmij(double a, double b){
13  if ((!std::isfinite(a))||(!std::isfinite(b))){
        errno = EDOM;
15      return 0;
    }
17  double c= a-b;
    if (!std::isfinite(c))
19      errno = ERANGE;
    return c;
21 }

23 double pomnoz(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b))){
25      errno = EDOM;
        return 0;
27  }
    double c= a*b;
29  if (!std::isfinite(c))
```

```
        errno = ERANGE;
    return c;
}

double podziel(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b))||(b==0)){
        errno = EDOM;
        return 0;
    }
    double c= a/b;
    if (!std::isfinite(c))
        errno = ERANGE;
    return c;
}
```

Listing 8.116. rozwiązanie zadania 7.6b

```
double dodaj(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b)))
        throw spoza_dziedziny();
    double c= a+b;
    if (!std::isfinite(c))
        throw wynik_spoza_zakresu();
    return c;
}

double odejmij(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b)))
        throw spoza_dziedziny();
    double c= a-b;
    if (!std::isfinite(c))
        throw wynik_spoza_zakresu();
    return c;
}

double pomnoz(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b)))
        throw spoza_dziedziny();
    double c= a*b;
    if (!std::isfinite(c))
        throw wynik_spoza_zakresu();
    return c;
}

double podziel(double a, double b){
    if ((!std::isfinite(a))||(!std::isfinite(b)))
        throw spoza_dziedziny();
    if (b==0)
        throw dzielenie_przez_zero();
    double c= a/b;
```

```
    if (!std::isfinite(c))
35      throw wynik_spoza_zakresu();
    return c;
37 }
```

Zadanie 7.7

Listing 8.117. rozwiązanie zadania 7.7 wersja oparta na wyjątkach

```
1 void rownanie(double a, double b, double c){
    try {
3     double delta = odejmij(pomnoz(b,b),pomnoz(4,
                                        pomnoz(a,c)));
5     double pierw_delta = pierw(delta);
      double x1 = podziel(dodaj(odejmij(0,b),pierw_delta),
7                                                 pomnoz(2,a));
      double x2 = podziel(odejmij(odejmij(0,b),pierw_delta),
9                                                  pomnoz(2,a));
      std::cout<<"x1 = "<<x1<<" x2 = "<<x2<<std::endl;
11  }
    catch(pierwiastek_z_ujemnej& e){
13      std::cout<<"Brak rozwiazan"<<std::endl;
    }
15  catch(wynik_spoza_zakresu){
        std::cout<<"Wyniki dzialan nie mieszcza sie w typie"
17                                                  <<std::endl;
    }
19  catch(spoza_dziedziny& e){
        std::cout<<"Argumenty dzialan spoza dziedziny"
21                                              <<std::endl;
    }
23  catch(...){
        std::cout<<"Nieznany blad"<<std::endl;
25  }
}
```

Listing 8.118. rozwiązanie zadania 7.7 wersja oparta na errno

```
void rownanie(double a, double b, double c){
2   errno = 0;
    double delta = odejmij(pomnoz(b,b),
4                      pomnoz(4,pomnoz(a,c)));
    if (delta>=0){
6       double pierw_delta = sqrt(delta);
        double x1 = podziel(dodaj(odejmij(0,b),
8                       pierw_delta),pomnoz(2,a));
        double x2 = podziel(odejmij(odejmij(0,b),
10                         pierw_delta),pomnoz(2,a));
            if (errno==0){
12              std::cout<<"x1 = "<<x1;
```

```
                if (delta>0)
14                  std::cout<<" x2 = "<<x2<<std::endl;
            }
16      }
        if ((errno==0)&&(delta<0))
18          std::cout<<"Brak rozwiazan"<<std::endl;
        if (errno==ERANGE)
20          std::cout<<"Wyniki dzialan nie mieszcza sie w typie"
                        <<std::endl;
22      else if (errno==EDOM)
            std::cout<<"Argumenty dzialan spoza dziedziny"
24                                                  <<std::endl;
        else if (errno>0)
26          std::cout<<"Nieznany blad"<<std::endl;
    }
```

Jednym z podstawowych kłopotów przy użyciu `errno`, jest fakt, że każda kolejna wykonywana operacja może nadpisać jego wartość. Oznacza to, że jeżeli dokładnie chcielibyśmy wyśledzić pierwotną przyczynę wystąpienia błędu, musielibyśmy sprawdzać wartość `errno` po wykonaniu każdej operacji. Użycie wyjątków pozbawione jest tej wady.

Zadanie 7.8

Listing 8.119. rozwiązanie zadania 7.8

```
1  void rownanie(double a, double b, double c){
       try {
3        double delta = odejmij(pomnoz(b,b),pomnoz(4,
                                    pomnoz(a,c)));
5        double pierw_delta = pierw(delta);
         double x1 = podziel(dodaj(odejmij(0,b),pierw_delta),
7                                                  pomnoz(2,a));
         double x2 = podziel(odejmij(odejmij(0,b),pierw_delta),
9                                                  pomnoz(2,a));
         std::cout<<"x1 = "<<x1<<" x2 = "<<x2<<std::endl;
11     }
       catch(pierwiastek_z_ujemnej& e){
13         std::cout<<"Brak rozwiazan"<<std::endl;
           throw;
15     }
       catch(wynik_spoza_zakresu& e){
17         std::cout<<"Wyniki dzialan nie mieszcza sie w typie"
                                                  <<std::endl;
19         throw;
       }
21     catch(spoza_dziedziny& e){
           std::cout<<"Argumenty dzialan spoza dziedziny"
23                                                 <<std::endl;
```

```
            throw;
        }
        catch(...){
            std::cout<<"Nieznany blad"<<std::endl;
            throw;
        }
    }
```

Zadanie 7.17

Listing 8.120. rozwiązanie zadania 7.17

```
    void bezpieczna() throw(){
        std::cout<<"jestem bezpieczna"<<std::endl;
    }
```

Zadanie 7.18

Listing 8.121. rozwiązanie zadania 7.18

```
int podziel2(int a, int b) throw(dzielenie_przez_zero){
    if (b==0)
        throw dzielenie_przez_zero();
    return a/b;
}
```

Zadania 7.19–7.20

Listing 8.122. rozwiązanie zadań 7.19–7.20

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include "wyjatki.h"

int podziel(int a, int b) throw(dzielenie_przez_zero,
                                 inny_blad){
    if (b==0)
        throw dzielenie_przez_zero();
    return a/b;
}

void unexpected(){
    std::cout<<"nieznany blad przy dzieleniu"
                                 <<std::endl;
            throw inny_blad();
}

void terminate(){
    std::cout<<"niezlapany wyjatek"<<std::endl;
```

```
21      exit(1);
    }
23
    int main(){
25      std::set_unexpected(unexpected);
        std::set_terminate(terminate);
27      std::vector<std::pair<int,int> > v(10);

29      for(int i=0;i<10;i++)
            std::cin>>v[i].first>>v[i].second;
31
        for(int i=0;i<10;i++) try {
33          std::cout<<podziel(v[i].first,v[i].second)
                                                <<std::endl;
35      }
        catch(...){
37      }
    }
```

Zakładamy, że dołączony w nagłówku programu plik `wyjatki.h` zawiera rozwiązanie zadania 7.22.

Zadanie 7.18

Listing 8.123. rozwiązanie zadania 7.22

```
    #include <iostream>
2   #include <vector>
    #include <csetjmp>
4
    jmp_buf bufor;
6
    int podziel(int a, int b){
8       if (b==0)
            longjmp(bufor,1);
10      return a/b;
    }
12  int main(){
        std::vector<std::pair<int,int> > v(10);
14      for(int i=0;i<10;i++){
            std::cin>>v[i].first>>v[i].second;
16      }
        for(int i=0;i<10;i++) {
18          if (setjmp(bufor)!=0){
            }
20          else
              std::cout<<podziel(v[i].first,v[i].second)
22                                                <<std::endl;
      }
24  }
```

Bibliografia

[1] Marshall Cline, *C++ FAQ — Frequently Asked Questions*, http://www.parashift.com/c++-faq-lite/
[2] Bruce Eckel, *Thinking in C++*, Helion, Warszawa 2002
[3] Jerzy Grębosz, *Symfonia C++ standard*, Editions, Kraków 2008
[4] Bjarne Stroustrup,*Język C++*, Wydawnictwa Naukowo-Techniczne, Warszawa 2002
[5] Standard C++03x, *ISO/IEC 14882:2003*, ISO 2003
[6] Standard C++11x, *ISO/IEC 14882:2011*, ISO 011
[7] *C++ reference*, http://en.cppreference.com/w/cpp